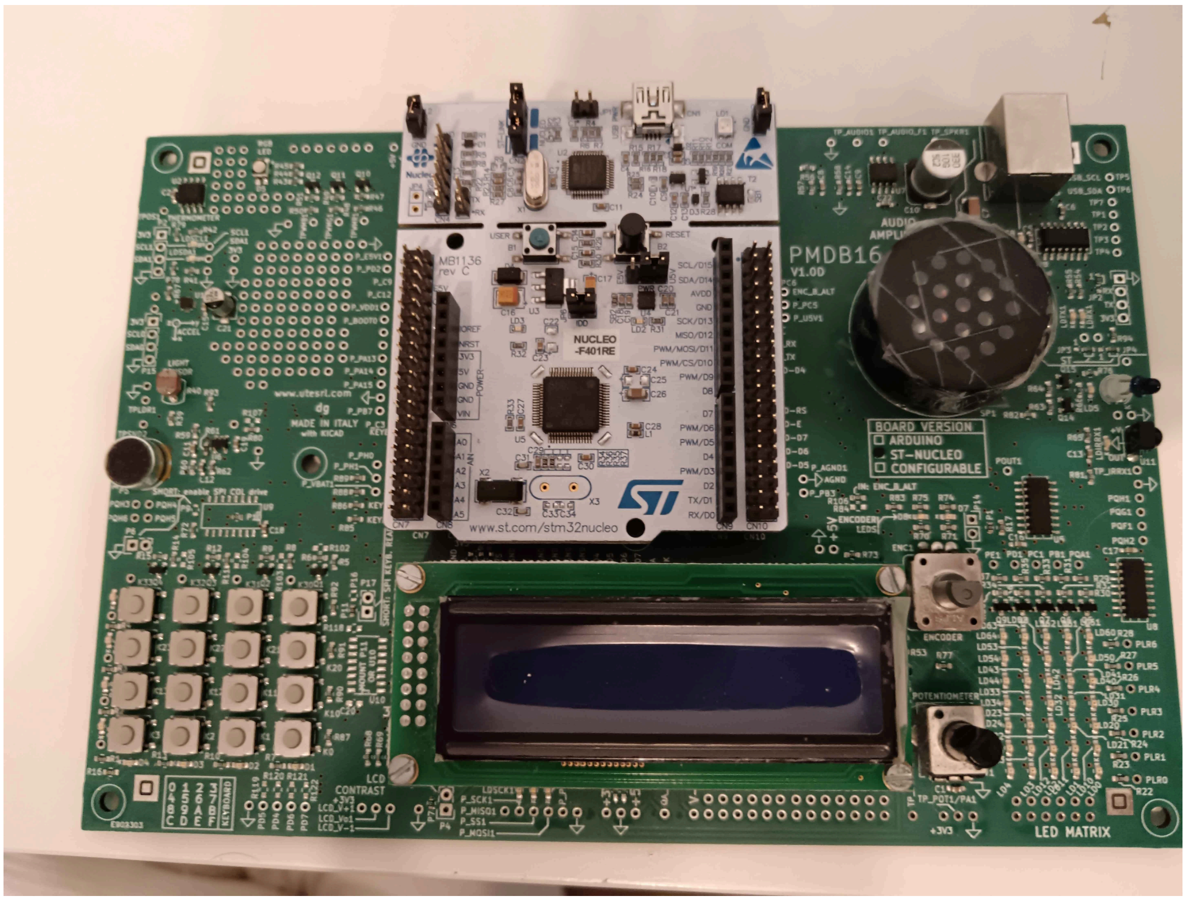
Computer Science Team B4

Sensor system

Homework Report



HOMEWORK NUMBER 5

2024 - 2025

POLITECNICO
MILANO 1863

Summary

I. Members work

Members	Exercise 1	Exercise 2	Exercise 3	General
Cazin Némo	✓	✓	✓	Wrote the report
Adrien Paliferro	✓	✓	✓	Did the exercises and participated in writing the report.
Heikki Leveelahti	✓	✓	✓	Reviewed the report
Osmo Kieksi	✓	✓	✓	
Constantijn Coppers	✓	✓	✓	

Github Link : **https://github.com/nemocazin/sensor_system_labs**

II. Exercise 1

General functionality:
In this project, we send a voltage value, collected by an ADC, to a terminal by using USART.

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**
ADC	PA1	ADC1_IN1
USART_TX	PA2	Transmit
USART_RX	PA3	Receive

2. **Board configuration**

First, we need to configure the ADC.
We put PA1, which is the potentiometer pin in the configuration "ADC1_IN1".
Because of this PIN Configuration, we need to configure the ADC channel 1.
We just need to change the parameter "Sampling Time" to 3 cycles to 480 cycles.
We also need to activate on the NVIC tab, the interruption of the ADC.

We want to send the value acquired on a terminal, that's why we need to configure the USART.
The configuration of USART is by default, we have the principal parameters :
- Mode : Asynchronous
- RS232 : Disable
- Baud Rate : 115200 Bits/s

We also need to make the conversions at a rate of 1Hz.
So we need to configure the timer TIM3, we have these principal parameters :
- Clock Source : Internal Clock
- Prescaler : 8399
- Period : 9999
- Trigger Event Selection : Update Event

Moreover, we need to activate the interruption on the NVIC Tab.

Finally, we can activate the floats in the properties tab of the project for a proper conversion of the float tension value.

After the general configuration, we can work on the code in the main.c file.

3. Code implementation

So first we need to activate the timer in the main function with this line :

```
/** Start the timer 3 **/
if(HAL_TIM_Base_Start_IT(&htim3) != HAL_OK)
{
  Error_Handler();
}
```

The timer is now working, so in the callback we can call the ADC callback :

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    // Check if it's TIM3 that caused the interrupt
    if (htim->Instance == TIM3)
    {
          if (HAL_ADC_Start_IT(&hadc1) != HAL_OK) // Start the ADC
conversion
          {
                 Error_Handler();
          }
    }
}
```

And so, for the ADC callback, we get the voltage value and we put it in string to send it via USART
Here is the ADC Callback function :

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
	uint32_t ADC_value = HAL_ADC_GetValue(hadc); // Get the value
of the potentiometer
	char message [64];
	int length = snprintf(message, sizeof(message), "Voltage :
%.2fV\r\n", (VOLTAGE*ADC_value)/RESOLUTION);
	HAL_UART_Transmit(&huart2, (uint8_t*)message, length, 100);
// Send ON USART the value
}
```

4. Another method

In the .ioc file, in Timers, we configure TIM3 as the previous method :
- Clock Source : Internal Clock,
- Prescaler : 8399
- Counter Period : 9999,

But this time, we go directly configure ADC1 in the .ioc file :
- We enable interruption in the NVIC settings
- We select Timer 3 Trigger Out event as an "External Trigger Conversion" in parameters settings
- then select Trigger detection on the rising edge.

This way, every time the timer triggers an interruption, the ADC receives a signal to start a conversion.

Upon completing the conversion, the ADC generates an End of Conversion (EOC) interrupt, which calls the HAL_ADC_ConvCpltCallback() function.
So we have in the initialization of the ADC these 2 new lines :

```
hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
// Trigger on rising edge
 hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO; //
ADC triggered by Timer 3
```

Moreover, in the code, we have to initialize the timer and the ADC in the main function :

```
/* Start of the Timer */
HAL_TIM_Base_Start(&htim3);  // Start timer 3

/* Start of ADC with interrupt mode */
HAL_ADC_Start_IT(&hadc1);
```

And then, we can use the ADC callback function without the timer callback function :

```
/* ADC Conversion callback function */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    if (hadc->Instance == ADC1)
    {
        adc_value = HAL_ADC_GetValue(hadc);
        float voltage = (adc_value / 4095.0) * 3.3;
        char buffer[64];
        int length = snprintf(buffer, sizeof(buffer), "Voltage:
%.2f V\r\n", voltage);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, length, 100);
    }
}
```

5. Execution

Link to the video : [Video of Homework_5_a]

III. Exercise 2

General functionality:
In this project, we write a voltage value, collected by an ADC, to a LCD screen.

Implementation:

1. **PIN lookup**

Pins are initialized as followed :

Peripheral	PIN	I/O
ADC	PA1	ADC1_IN1
USART_TX	PA2	Transmit
USART_RX	PA3	Receive
LCD_BL_ON	PA4	GPIO output
LCD_E	PB1	GPIO output
LCD_RS	PB2	GPIO output
LCD_D4	PB12	GPIO output
LCD_D5	PB13	GPIO output
LCD_D6	PB14	GPIO output
LCD_D7	PB15	GPIO output

2. **Board configuration**

The configuration is almost the same as the exercise 1.
First, we need to configure the ADC.
We put PA1, which is the potentiometer pin in the configuration "ADC1_IN1".
Because of this PIN Configuration, we need to configure the ADC channel 1.
We just need to change the parameter "Sampling Time" to 3 cycles to 480 cycles.
We also need to activate on the NVIC tab, the interruption of the ADC.

We want to send to value acquired on a terminal, that's why we need to configure the USART.
The configuration of USART is by default, we have the principal parameters :

- Mode : Asynchronous
- RS232 : Disable
- Baud Rate : 115200 Bits/s

We also need to make the conversions at a rate of 1Hz.
So we need to configure the timer TIM3, we have these principal parameters :

- Clock Source : Internal Clock
- Prescaler : 8399
- Period : 9999
- Trigger Event Selection : Update Event

We also need to activate the interruption on the NVIC Tab.

Finally, we need to config the pins of the LCD Display, so we use :

- LCD_BL_ON (GPIO_Output) = PA4
- LCD_E (GPIO_Output) = PB1
- LCD_RS (GPIO_Output) = PB2
- LCD_D4 (GPIO_Output) = PB12
- LCD_D5 (GPIO_Output) = PB13
- LCD_D6 (GPIO_Output) = PB14
- LCD_D7 (GPIO_Output) = PB15

After the configuration, we add the “PMDB16_LCD.c” and the “PMDB16_LCD.h” files to the "project/Core/Src" folder.
We now can add “#include "PMDB16_LCD.h“ in our main.c file.

We can activate the floats in the properties tab of the project for a proper conversion of the float tension value.

3. **Code implementation**

We can now go into the main.c file and use the same code as in the exercise 1
We also include the LCD files and initialize the LCD with theses 3 lines :

```
lcd_initialize();     // Initialize the LCD controller
lcd_backlight_ON();   // Turn ON the LCD backlight
lcd_clear();          // Clear the LCD
```

Moreover, in the ADC callback, we replace the USART transmission by the LCD functions.
And so we have this callback function :

```
/**
  * @brief ADC Callback Function
  *
  * @param hadc The ADC
  *
  * @details Is called in the timer callback
  */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
       uint32_t ADC_value = HAL_ADC_GetValue(hadc); // Get the value of the
potentiometer
       char message [64];
       int length = snprintf(message, sizeof(message), "Voltage : %.2fV\r\n",
(VOLTAGE * ADC_value) / RESOLUTION); // Put value in the message variable
       lcd_println(message, TOP_ROW); // Put the message variable on top row of
the LCD
       lcd_drawBar((MAX_PIXEL_ROW * ADC_value) / RESOLUTION); // Put a variable
bar on the bottom row of the LCD
}
```

4. Execution

Link to the video : Video of Homework_5_b

IV. Exercise 3

General functionality:
In this project, we send from the PC via UART a string of variable length that is displayed on the LCD.

Implementation:

1. **PIN lookup**

Peripheral	**PIN**	**I/O**
USART_TX	PA2	Transmit
USART_RX	PA3	Receive
LCD_BL_ON	PA4	GPIO output
LCD_E	PB1	GPIO output
LCD_RS	PB2	GPIO output
LCD_D4	PB12	GPIO output
LCD_D5	PB13	GPIO output
LCD_D6	PB14	GPIO output
LCD_D7	PB15	GPIO output

2. **Board configuration**

We want to receive a string sent on a terminal, that's why we need to configure the USART.
The configuration of USART is by default, we have the principal parameters :

- Mode : Asynchronous
- RS232 : Disable
- Baud Rate : 115200 Bits/s

Finally, we need to config the pins of the LCD Display, so we use :

- LCD_BL_ON (GPIO_Output) = PA4
- LCD_E (GPIO_Output) = PB1
- LCD_RS (GPIO_Output) = PB2
- LCD_D4 (GPIO_Output) = PB12
- LCD_D5 (GPIO_Output) = PB13
- LCD_D6 (GPIO_Output) = PB14

- LCD_D7 (GPIO_Output) = PB15

After the configuration, we add the “PMDB16_LCD.c” and the “PMDB16_LCD.h” files to the "project/Core/Src" folder.
We now can add “#include "PMDB16_LCD.h“ in our main.c file.

3. Code implementation

First, we need to include the LCD files and initialize the LCD with theses 3 lines :

```
lcd_initialize();     // Initialize the LCD controller
lcd_backlight_ON();   // Turn ON the LCD backlight
lcd_clear();          // Clear the LCD
```

After that we need to get the string that we write.
For that, we need to initialize a variable for the string with a size of 16 characters.

```
char rxBuffer[SIZE_ROW];
```

Then, we will get every characters written until there is an end of line character (\r or \n)
We add a null character \0 to indicate the end of the string.
So we got this function :

```
/**
 * @brief Get characters for the string
 */
void receiveString() {
    uint8_t index = 0; // buffer index
    char ch;
    // Get characters
    while (index < SIZE_ROW) {
        if (HAL_UART_Receive(&huart2, (uint8_t*)&ch, 1, 1000) == HAL_OK) {
            if (ch == '\n' || ch == '\r') {
                break; // Break if end of line
            }
            rxBuffer[index++] = ch; // Add Character to buffer
        }
    }
    rxBuffer[index] = '\0'; // Add null character at the end of the string
}
```

We now can display the string on the LCD display with the function called in the main while loop

```
/**
 * @brief Get and display the string
 */
void displayString() {
    receiveString(); // Get char string
    lcd_println(rxBuffer, TOP_ROW); // Print on LCD
}
```

4. Execution

Link to the video : [Video of Homework_5_c]

V. Professors comments

EXERCISE 1 :
EXERCISE 2 :
EXERCISE 3 :