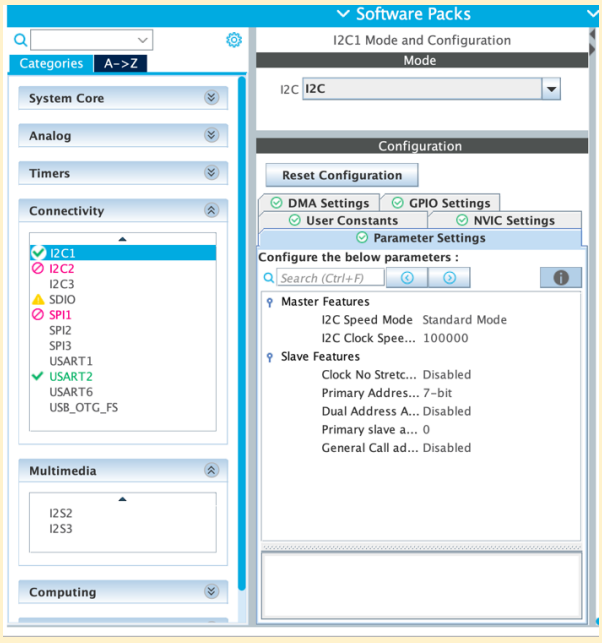
Mark	/11

Team name:	*A1*		
Homework number:	*HOMEWORK 07*		
Due date:	05/11/24		
Contribution	NO	Partial	Full
Piombo			*x*
Fumagalli			*x*
Pierfederici			*x*
Zenoni			*x*
Ferraro			*x*
Notes:			

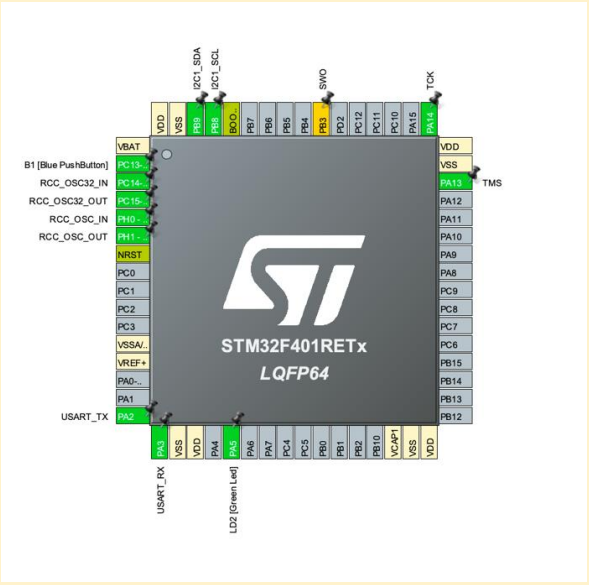
Project name	Temperature Sensor		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

Part 1:

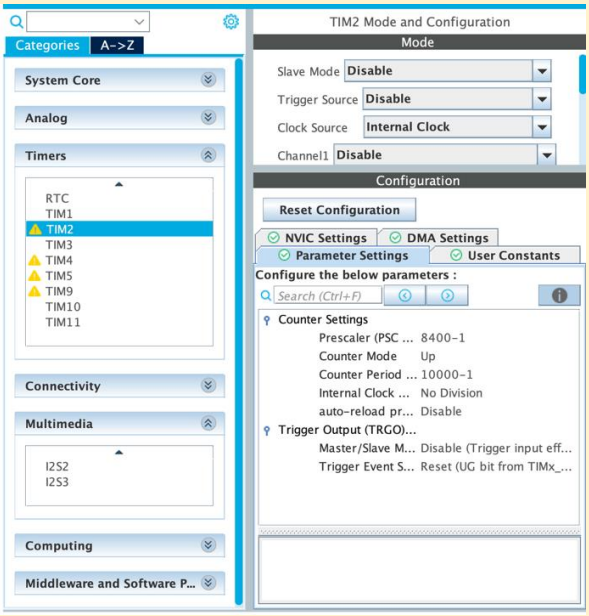
Starting from the ".ioc" we enabled the I2C1 and we set the "I2C Clock Speed" to 100KHz.



Then, we configured our pin of interest (PB8 - SCL, PB9 - SDA) to communicate with the I2C to the temperature 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 temperature data from the sensor each second.



We configured the UART in the DMA mode to send the temperature to the pc.

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

```
54
55  /* USER CODE BEGIN PV */
56
57  uint8_t thermo_address = 0b10010000;    //address of the LM75 (thermometer) peripheral - left shifted by 1
58  uint8_t temp_reg_pointer = 0b00000000;  //internal address of the temperature register of the LM75
59  uint8_t rx_bytes[2]= {0 ,0};            //save the 2 bytes received from the LM75 temp sensor
60  int16_t rx_temperature = 0;             //save both temperature bytes (MSB and LSB)
61  float tx_temperature = 0;               //save converted temperature, ready to be sent
62
63  int length;
64  char string[STR_LEN];
65
66  /* USER CODE END PV */
67
```

In the main function we only set the Pointer Register of the LM75 with the Temperature Register Address and we started TIM2 in interrupt mode.

```
132
133     //set the pointer register in order to read temp register (+0 to WRITE LM75)
134     HAL_I2C_Master_Transmit(&hi2c1, thermo_address+0, &temp_reg_pointer, 1, 10);
135
136     HAL_TIM_Base_Start_IT(&htim2);    //start the timer
137
138     /* USER CODE END 2 */
139
```

In the TIM2 Callback we checked if the communication succeeded and then we converted the binary value in a float with the degree value. In the end we sent the temperature using UART with DMA.

```
79 /* Private user code ---------------------------------------------------------*/
80 /* USER CODE BEGIN 0 */
81
82 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim) {
83     if (htim == &htim2) {
84         //read 2 bytes of the temperature register (+1 to READ)
85         if (HAL_I2C_Master_Receive(&hi2c1, thermo_address+1, rx_bytes, 2, 10) == HAL_OK) {
86             rx_temperature = (rx_bytes[0] << 8) + rx_bytes[1];    //save MSB (shifted by 8) and LSB
87             tx_temperature = rx_temperature/256.0;               //temperature conversion: division by 2^8
88             length = snprintf(string, sizeof(string), "TEMPERATURE: %.3f°C\n", tx_temperature);
89         } else {
90             length = snprintf(string, sizeof(string), "ERROR reading from temperature sensor!\n");
91         }
92         HAL_UART_Transmit_DMA(&huart2, string, length);
93     }
94 }
95
96 /* USER CODE END 0 */
97
```

All our boards are equipped with the LM75 sensor, thus we didn't encounter the bug.

Anyway, we found out the reason reading the datasheets of the 2 sensors: the LM75 stops the conversion while we are reading it, the LM75B instead doesn't interrupt the conversion, so it can happen that we save MSB and LSB not belonging to the same conversion.

Figure 1. LM75

7.1 Overview

The LM75A temperature sensor incorporates a band-gap type temperature sensor and 9-bit ADC (sigma-delta ADC). The temperature data output of the LM75A is available at all times via the I^2C bus. If a conversion is in progress, it will be stopped and restarted after the read. A digital comparator is also incorporated that compares a series of readings, the number of which is user-selectable, to user-programmable setpoint and hysteresis values. The comparator trips the O.S. output line, which is programmable for mode and polarity. The LM75A has an integrated low-pass filter on both the SDA and the SCL line. These filters increase communications reliability in noisy environments.

Figure 2. LM75B

The temperature register always stores an 11-bit two's complement data giving a temperature resolution of 0.125 °C. This high temperature resolution is particularly useful in applications of measuring precisely the thermal drift or runaway. When the LM75B is accessed the conversion in process is not interrupted (that is, the I^2C-bus section is totally independent of the Sigma-Delta converter section) and accessing the LM75B continuously without waiting at least one conversion time between communications will not prevent the device from updating the Temp register with a new conversion result. The new conversion result will be available immediately after the Temp register is updated.

A possible way to solve this problem is trying to receive temperature values in a row and compare them in order to decide if the received data is valid.