INTERFACING HTU21D(F)(TEMPERATURE AND HUMIDITY SENSOR) WITH STM32F401RE

MINI PROJECT REPORT

by

ABARNA B.

20068161

DESHILA TECHNOLOGY RESEARCH INSTITUTE

MYSORE – 570006

&

SKANRAY TECHNOLOGIES PVT. LTD.

MYSORE - 570016

October 2023

1. INTRODUCTION

1.1 HTU21D(F) SENSOR:

The HTU21D(F) sensor is a digital humidity and temperature sensor designed for various applications, including environmental monitoring, weather stations, industrial control systems, and more. It is manufactured by TE Connectivity, a company that specializes in sensor and connectivity solutions. The "F" in HTU21DF indicates that it is a small, surface-mount device.

1.1.2 KEY FEATURES OF THE HTU21D(F) SENSOR

a) **Humidity Sensing:** The sensor can accurately measure relative humidity (RH) in the range of 0% to 100%. It is capable of providing precise humidity measurements, making it suitable for applications where humidity control is important.

b) **Temperature Sensing:** The HTU21D(F) can measure temperature in the range of -40°C to 125°C. It offers high accuracy in temperature measurements.

c) **Digital Interface**: This sensor communicates with a microcontroller or other digital devices using an I2C (Inter-Integrated Circuit) interface. This makes it easy to integrate into various projects.

d) **Low Power Consumption:** The sensor is designed to operate with low power consumption, which is important for battery-powered and energy-efficient applications.

e) **Calibrated Sensor:** The HTU21D(F) is pre-calibrated, which means it can be started without the need for extensive calibration procedures.

f) **Compact Size:** The sensor is typically small and compact, making it suitable for applications with space constraints.

g) **Durable Design:** It is often built with protective features to make it robust and reliable in various environmental conditions.

h) **Wide Range of Applications:** The HTU21D(F) sensor can be used in a variety of applications, including meteorological instruments, HVAC systems, data loggers, and any other applications where accurate humidity and temperature measurements are required.

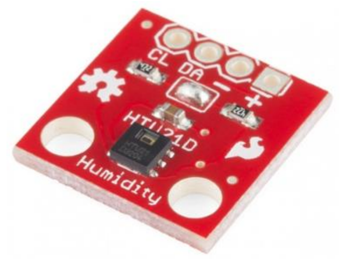


Fig. 1 HTU21D(F) Sensor

1.1.3 PIN CONFIGURATION

The HTU21D(F) sensor has a relatively simple pin configuration. It typically has four pins: VDD, GND, SDA, and SCL.

a) **VDD (Voltage Supply):** This is the power supply pin for the sensor. It is connected to a voltage source typically in the range of 2.1V to 3.6V. The sensor operates on this supply voltage.

b) **GND (Ground):** This is the ground connection for the sensor. It should be connected to the ground (0V) reference of your power supply.

c) **SDA (Serial Data):** This is the data line for the I2C (Inter-Integrated Circuit) communication protocol. It is used for bidirectional data transfer between the sensor and the microcontroller. You connect this pin to the SDA pin of your microcontroller or I2C bus.

d) **SCL (Serial Clock):** This is the clock line for the I2C communication. It provides the clock signal for synchronized data transfer between the sensor and the microcontroller. Connect this pin to the SCL pin of your microcontroller or I2C bus.

1.1.4 SENSOR PERFORMANCE

a. Relative Humidity

TABLE I: Relative Humidity.

Characteristics		**Symbol**	**Min**	**Typ**	**Max**	**Unit**
Resolution	12 bits			0.04		%RH
	8 bits			0.7		%RH
Humidity Operating Range		RH	0		100	%RH
Relative Humidity	typ			±2		%RH

Accuracy @25°C (20%RH to 80%RH)	max		See graph 1			%RH
Replacement		fully interchangeable				
Temperature coefficient (from 0°C to 80°C)		Tcc			-0.15	%RH/°C
Humidity Hysteresis				±1		%RH
Measuring Time [(1)]	12 bits			14	16	ms
	11 bits			7	8	ms
	10 bits			4	5	ms
	8 bits			2	3	ms
PSRR					±10	LSB
Recovery time after 150 hours of condensation		t		10		s
Long term drift				0.5		%RH/yr
Response Time (at 63% of signal) from 33 to 75%RH [(2)]		τRH		5	10	s

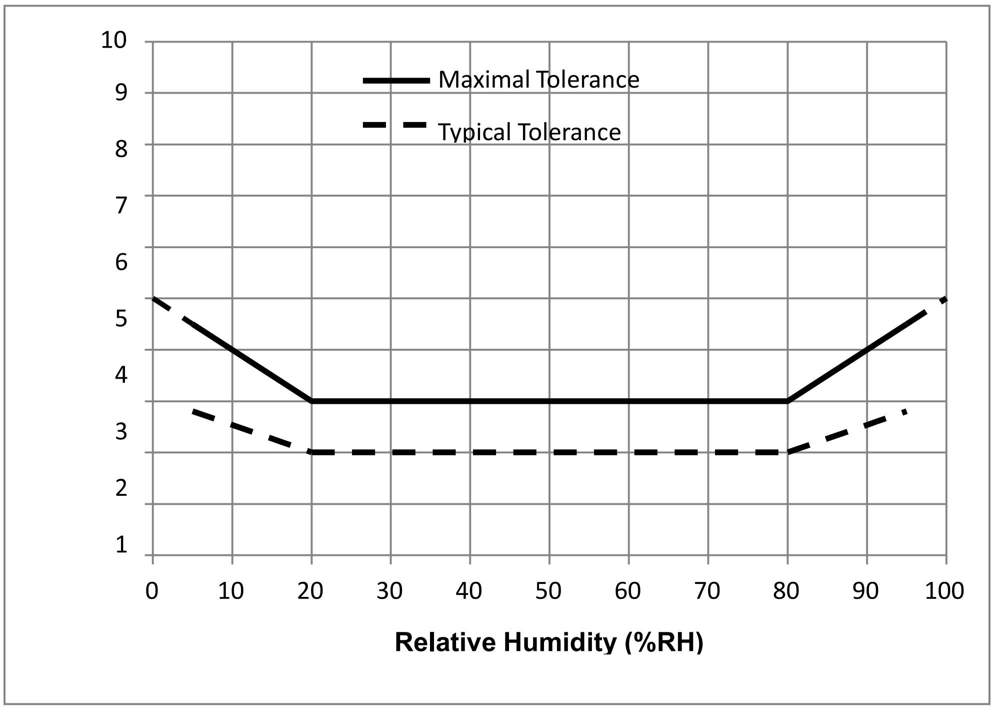


Fig. 2 Relative Humidity Error Budget Conditions at 25°C

b. Relative Temperature

TABLE II: Relative Temperature.

Characteristics		Symbol	Min	Typ	Max	Unit
Resolution	14 bit			0.01		°C
	12 bit			0.04		°C
Temperature Operating Range		T	-40		+125	°C
Temperature Accuracy @25°C	typ			±0.3		°C
	max		See graph 2			°C
Replacement			fully interchangeable			
Measuring time [(1)]	14 bit			44	50	ms
	13 bit			22	25	ms
	12 bit			11	13	ms
	11 bit			6	7	ms
PSSR					±25	LSB
Long term drift				0.04		°C/yr
Response Time (at 63% of signal) from 15°C to 45°C [(2)]		τT		10		s

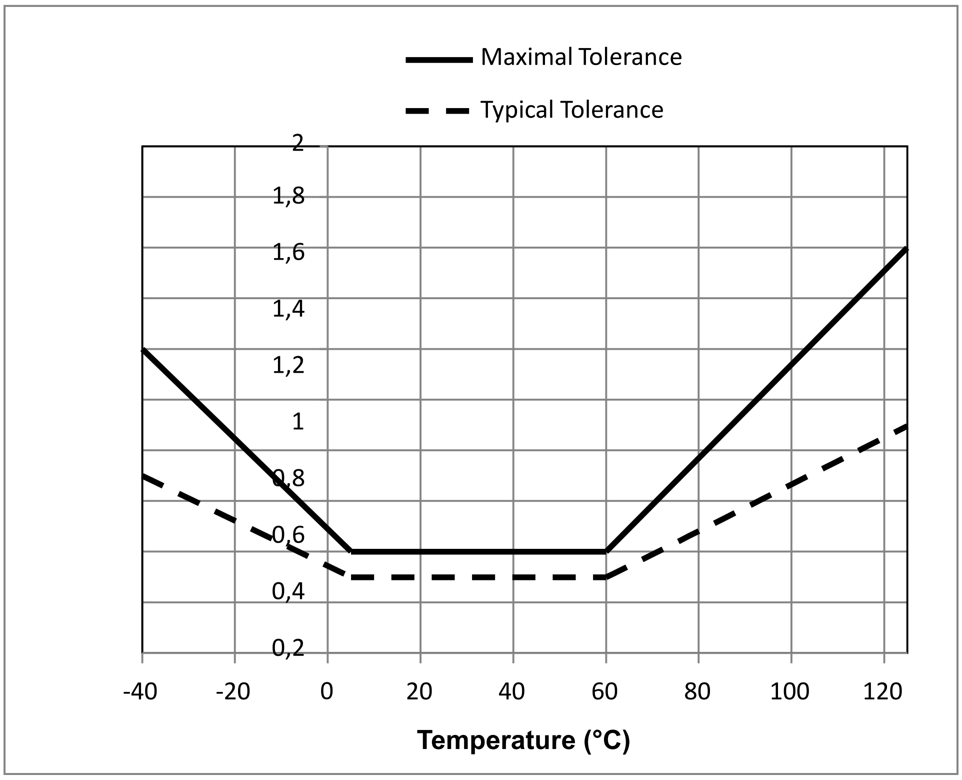


Fig. 3 Temperature Error Budget

1.2 STM32F401RETX MICROCONTROLLER

The STM32F401RETX is a microcontroller from STMicroelectronics, based on the ARM Cortex-M4 core. It is part of the STM32F4 series of microcontrollers, which are known for their high performance and versatility.

a) **Core:** It is based on the ARM Cortex-M4 processor core, which is a 32-bit RISC core with hardware floating-point support. The Cortex-M4 core provides high performance and low power consumption.
b) **Clock Speed:** The STM32F401RETX typically operates at a maximum clock speed of 84 MHz.
c) **Flash Memory:** It features up to 512 KB of Flash memory for program storage.
d) **RAM:** It has up to 96 KB of SRAM for data storage.
e) **Peripherals:** The microcontroller is equipped with a wide range of peripherals, including GPIO pins, timers, UART, SPI, I2C, and more. These peripherals can be used for various applications and interfacing with external devices.
f) **Analog-to-Digital Converter (ADC):** It has a 12-bit ADC with multiple channels, allowing for analog signal conversion.
g) **Communication Interfaces:** The STM32F401RETX supports various communication interfaces, including SPI, I2C, USART, USB, and CAN, making it suitable for a wide range of applications.
h) **Operating Voltage:** It typically operates at a supply voltage of 1.7V to 3.6V.
i) **Integrated Development Environment (IDE)**: STMicroelectronics provides the STM32CubeIDE and STM32CubeMX tools for development, which can help in configuring and programming the microcontroller.
j) **Power Modes:** The microcontroller supports different power-saving modes to reduce power consumption when idle or not in use, making it suitable for battery-powered applications.
k) **Security:** Some models within the STM32F4 series have hardware security features, such as a unique device ID and cryptographic accelerators, which can be used to enhance the security of embedded applications.
l) **Operating Temperature:** It is designed to operate within a specified temperature range, typically from -40°C to +85°C or wider, depending on the specific model.

1.2.1 PIN CONFIGURATION (PINS CONNECTED WITH HTU21D(F) SENSOR)

a) **VDD:** Connect this to a 3.3V power supply, which matches the voltage supply of the sensor.

b) **GND:** Connect this to the ground (0V) of the power supply and to the HTU21D(F) sensor's ground (GND) pin.

c) **SDA (I2C Data):** Connect this to the SDA (Serial Data) pin of the HTU21D(F) sensor.

d) **SCL (I2C Clock):** Connect this to the SCL (Serial Clock) pin of the HTU21D(F) sensor.

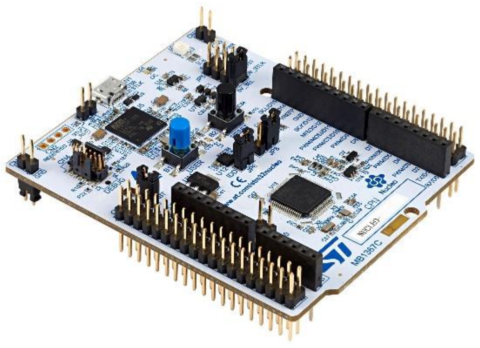


Fig. 4 STM32F401RE Microcontroller

2. INTERFACING AND READING THE TEMPERATURE AND HUMIDITY

2.1 Hardware Connections:

a. Power Supply:

i. Connect the VDD pin of the HTU21D(F) sensor to a 3.3V power supply.

ii. Connect the GND pin of the HTU21D(F) sensor to the ground (0V) of the power supply.

iii. Connect the VDD pin of the STM32F401RETX to a 3.3V power supply.

iv. Connect the GND pin of the STM32F401RETX to the ground (0V) of the power supply.

b. I2C Interface:

i. Connect the SDA (Serial Data) pin of the HTU21D(F) sensor to the SDA (I2C data) pin of the STM32F401RETX.

ii. Connect the SCL (Serial Clock) pin of the HTU21D(F) sensor to the SCL (I2C clock) pin of the STM32F401RETX.

2.2 Software Configuration (Using STM32CubeIDE):

i. Create a new STM32CubeIDE project for your STM32F401RETX microcontroller.

ii. Configure the project settings as needed, including selecting the target STM32 microcontroller.

iii. In STM32CubeIDE, open the "Pinout & Configuration" tab.

iv. Enable the I2C interface that you plan to use (I2C1, I2C2, etc.). Configure the pins as SDA and SCL.

v. Set the appropriate I2C peripheral frequency.

vi. Save the Pinout configuration.

vii. In STM32CubeIDE, go to the "Peripherals" tab.

viii. Click on "I2C" and configure the I2C settings as needed (e.g., clock speed, addressing).

ix. Save the configuration.

x. Write C code to initialize the I2C communication with the HTU21D sensor. This code should include initializing the I2C peripheral, setting up the communication parameters, and enabling the I2C interface.

xi. Write code to read humidity and temperature data from the HTU21D sensor using I2C communication.

xii. Use a UART (serial communication) interface to transmit the data to the serial monitor.

xiii. Compile your code in STM32CubeIDE.

xiv. Flash the compiled code onto your STM32F401RETX microcontroller.

xv. Power up your hardware setup.

xvi. Connect the microcontroller to a computer running a serial terminal Tera Term to view the temperature and humidity data transmitted via UART.

2.3 Coding Part

```
/* USER CODE BEGIN Header */
/**
*************************************************************************
******
* @file           : main.c
* @brief          : Main program body
*************************************************************************
******
@attention
*
Copyright (c) 2023 STMicroelectronics.
```

```
All rights reserved.
*
This software is licensed under terms that can be found in the LICENSE
file
in the root directory of this software component.
If no LICENSE file comes with this software, it is provided AS-IS.
*
************************************************************************
******
*/
/* USER CODE END Header */
/* Includes ------------------------------------------------------------
------*/
#include "main.h"

/* Private includes ----------------------------------------------------
------*/
/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------
------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------
------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------
------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------
------*/
I2C_HandleTypeDef hi2c1;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------
------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2C1_Init(void);
/* USER CODE BEGIN PFP */
void print(int val);
```

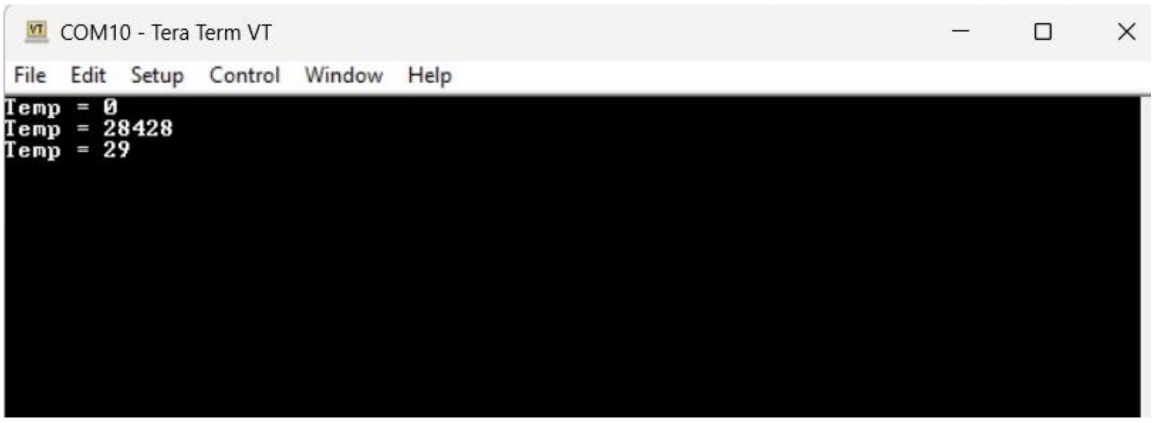
```
/* USER CODE END PFP */

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

/* USER CODE END 0 */

/**
@brief  The application entry point.
@retval int
*/
int main(void) {
/* USER CODE BEGIN 1 */

/* USER CODE END 1 */

/* MCU Configuration---------------------------------------------------
----*/

/* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
MX_I2C1_Init();
/* USER CODE BEGIN 2 */

uint8_t cmd = 0xE3, status, data_array[2] = { 0 };
uint16_t data = 0;
int temp, humidity;
HAL_Delay(2);

status = HAL_I2C_Master_Transmit(&hi2c1, 64 << 1, &cmd, 1, 1000);
print(status);
HAL_I2C_Master_Receive(&hi2c1, 64 << 1, data_array, 2, 1000);
data = ((data_array[0] << 8) | (data_array[1]));
print(data);
temp = (-46.85 + 175.72 * ((float) data / pow(2, 16)));
// humidity = (-6 + 125 * ((float) data / pow(2, 16)));
// print(humidity);
print(temp);

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
```

```
while (1) {
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}

/**
@brief System Clock Configuration
@retval None
*/
void SystemClock_Config(void) {
RCC_OscInitTypeDef RCC_OscInitStruct = { 0 };
RCC_ClkInitTypeDef RCC_ClkInitStruct = { 0 };

/** Configure the main internal regulator output voltage
*/
__HAL_RCC_PWR_CLK_ENABLE();
__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);

/** Initializes the RCC Oscillators according to the specified
parameters
in the RCC_OscInitTypeDef structure.
*/
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
RCC_OscInitStruct.HSIState = RCC_HSI_ON;
RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
RCC_OscInitStruct.PLL.PLLM = 16;
RCC_OscInitStruct.PLL.PLLN = 336;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
RCC_OscInitStruct.PLL.PLLQ = 7;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
Error_Handler();
}

/** Initializes the CPU, AHB and APB buses clocks
*/
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK
| RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
{
Error_Handler();
}
}

/**
@brief I2C1 Initialization Function
@param None
@retval None
*/
static void MX_I2C1_Init(void) {

/* USER CODE BEGIN I2C1_Init 0 */
/* USER CODE END I2C1_Init 0 */

/* USER CODE BEGIN I2C1_Init 1 */

/* USER CODE END I2C1_Init 1 */
hi2c1.Instance = I2C1;
hi2c1.Init.ClockSpeed = 100000;
hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
hi2c1.Init.OwnAddress1 = 0;
hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
hi2c1.Init.OwnAddress2 = 0;
hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
if (HAL_I2C_Init(&hi2c1) != HAL_OK) {
Error_Handler();
}
/* USER CODE BEGIN I2C1_Init 2 */

/* USER CODE END I2C1_Init 2 */

}

/**
@brief USART2 Initialization Function
@param None
@retval None
*/
static void MX_USART2_UART_Init(void) {

/* USER CODE BEGIN USART2_Init 0 */

/* USER CODE END USART2_Init 0 */

/* USER CODE BEGIN USART2_Init 1 */

/* USER CODE END USART2_Init 1 */
huart2.Instance = USART2;
huart2.Init.BaudRate = 115200;
huart2.Init.WordLength = UART_WORDLENGTH_8B;
huart2.Init.StopBits = UART_STOPBITS_1;
huart2.Init.Parity = UART_PARITY_NONE;
huart2.Init.Mode = UART_MODE_TX_RX;
huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart2.Init.OverSampling = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart2) != HAL_OK) {
Error_Handler();
}
/* USER CODE BEGIN USART2_Init 2 */

/* USER CODE END USART2_Init 2 */

}

/**
@brief GPIO Initialization Function
@param None
@retval None
*/
static void MX_GPIO_Init(void) {
```

```c
GPIO_InitTypeDef GPIO_InitStruct = { 0 };
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

/* GPIO Ports Clock Enable */
__HAL_RCC_GPIOC_CLK_ENABLE();
__HAL_RCC_GPIOH_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();
__HAL_RCC_GPIOB_CLK_ENABLE();

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin : B1_Pin */
GPIO_InitStruct.Pin = B1_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : LD2_Pin */
GPIO_InitStruct.Pin = LD2_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void print(int val) {
char str[20] = "";
sprintf(str, "Temp = %d\n", val);

HAL_UART_Transmit(&huart2, (uint8_t*) str, strlen(str), 1000);
}

/* USER CODE END 4 */

/**
@brief  This function is executed in case of error occurrence.
@retval None
*/
void Error_Handler(void) {
/* USER CODE BEGIN Error_Handler_Debug */
/* User can add his own implementation to report the HAL error return
state */
__disable_irq();
while (1) {
}
/* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
@brief  Reports the name of the source file and the source line number
where the assert_param error has occurred.
@param  file: pointer to the source file name
@param  line: assert_param error line source number
@retval None
```

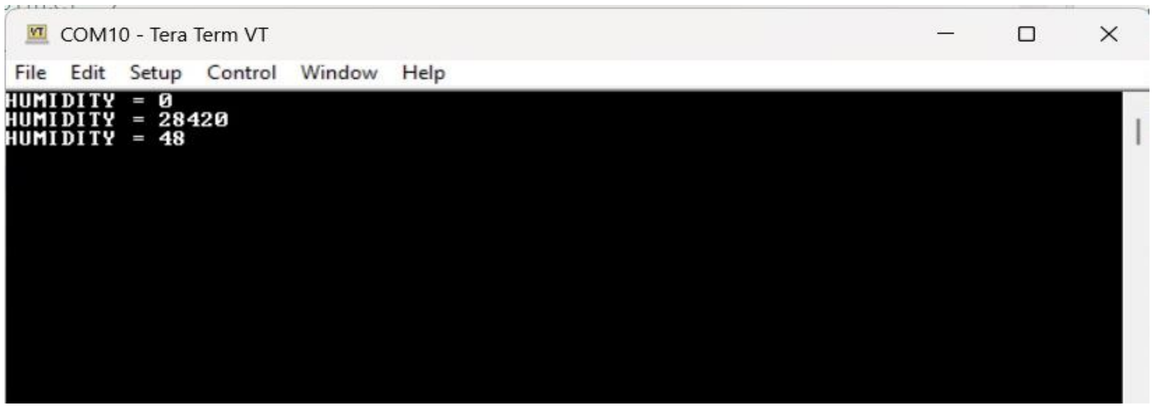
```
*/
void assert_failed(uint8_t *file, uint32_t line)
{
/* USER CODE BEGIN 6 */
/* User can add his own implementation to report the file name and line
number,
ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
/* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

2.4 Output Obtained



```
Temp = 0
Temp = 28428
Temp = 29
```

Fig. 5 Temperature Reading in Serial Monitor



```
HUMIDITY = 0
HUMIDITY = 28420
HUMIDITY = 48
```

Fig. 6 Humidity Reading in Serial Monitor