

# HTU21D(F) Sensor-Digital Relative Humidity sensor with Temperature output

## • Introduction

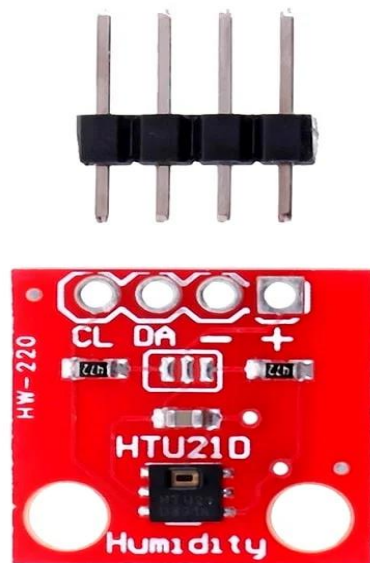
In today's world of advanced technology and automation, sensors play a pivotal role in gathering and transmitting data for various applications. Among these sensors, the digital relative humidity sensor with temperature output is a crucial tool, enabling accurate environmental monitoring in a wide range of industries. This two-in-one sensor measures both relative humidity (RH) and temperature, making it invaluable for applications such as weather forecasting, climate control in buildings, industrial processes, and even healthcare.

## • Features

1. Seamless interchangeability without the need for calibration under standard conditions.
2. Rapid desaturation, even after extended periods of saturation.
3. Suitable for integration into automated assembly processes, including those using Pb-free and reflow techniques.
4. Individually marked to meet rigorous traceability standards.

## • Applications

1. **Weather Monitoring:** These sensors are fundamental in weather stations and meteorological equipment, aiding in the prediction of weather conditions.
2. **HVAC Systems:** Heating, Ventilation, and Air Conditioning (HVAC) systems use these sensors to control indoor climate, ensuring comfort and energy efficiency.
3. **Industrial Processes:** Many industrial processes require strict control of temperature and humidity, such as in pharmaceutical and semiconductor manufacturing.
4. **Agriculture:** Precision agriculture relies on these sensors to monitor and control conditions in greenhouses and other agricultural environments.
5. **Food and Beverage Industry:** In food storage and processing, maintaining the right humidity and temperature is essential for preserving quality and safety.
6. **Museums and Archives:** These sensors help protect valuable artifacts and historical documents by maintaining optimal environmental conditions.
7. **Healthcare:** In healthcare, they play a role in monitoring conditions in laboratories, storage areas, and medical facilities.



**Fig: HTU21D Temperature and Humidity Sensor Module**

- **PERFORMANCE SPECS**

### **MAXIMUM RATINGS**

Ratings	Symbol	Value	Unit
Storage Temperature	$T_{stg}$	-40 to 125	$^{\circ}\text{C}$
Supply Voltage (Peak)	$V_{cc}$	3.8V	$V_{dc}$
Humidity Operating Range	RH	0 to 100	%RH
Temperature Operating Range	$T_a$	-40 to +125	$^{\circ}\text{C}$
VDD to GND		-0.3 to 3.6V	V
Digital I/O pins (DATA/SCK) to VDD		-0.3 to VDD+0.3	V
Input current on any pin		-10 to +10	mA

Peak conditions: less than 10% of the operating time

Exposure to absolute maximum rating conditions for extended periods may affect the sensor reliability.

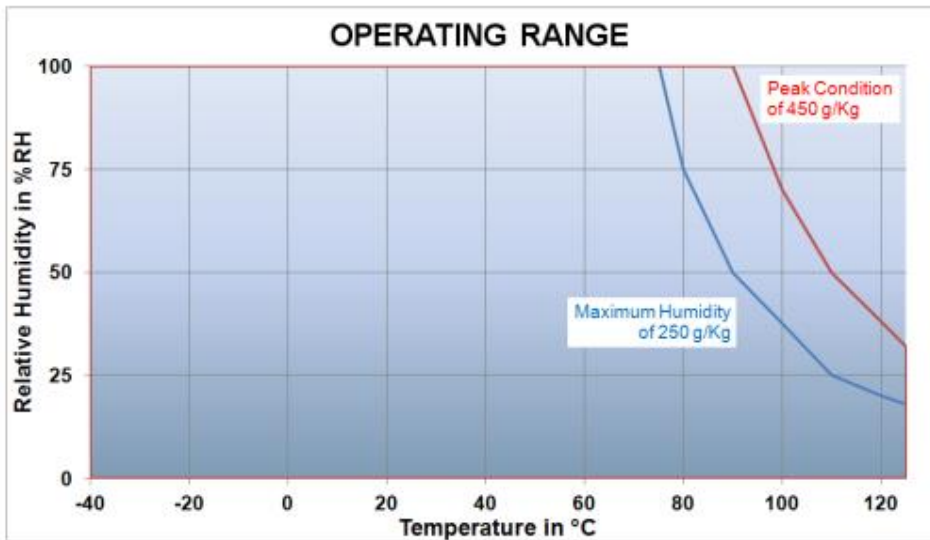


Fig: Operating range

In the realm of humidity and temperature measurement, the HTU21D(F) digital humidity sensors stand out as remarkable transducers tailored for OEM applications demanding unwavering precision and reliability. These sensors pave the way for seamless integration with microcontrollers, facilitating the acquisition of humidity and temperature data in a digital format. With their low power consumption and a design optimized for high-volume, cost-sensitive applications burdened by space constraints, the HTU21D(F) sensors are at the forefront of innovation.

Each HTU21D(F) sensor undergoes individual calibration and rigorous testing, assuring impeccable performance. Lot identification information is etched onto the sensor, and an electronic identification code is securely stored within the chip, which can be effortlessly retrieved through a simple command. Furthermore, these sensors possess the capability to detect low battery levels, enhancing operational dependability. To top it off, the sensors' resolution can be customized through a command interface, allowing for a wide range of options, from 8/12bit to an impressive 12/14bit for humidity and temperature measurements.

- **Enhancing Precision and Reliability: HTU21D(F) Digital Humidity Sensors**

The HTU21D(F) digital humidity sensors represent a groundbreaking solution for those seeking unparalleled accuracy and reliability in humidity and temperature measurement. These sensors are engineered for seamless connectivity with microcontrollers, simplifying the process of obtaining digital humidity and temperature data. Their design, with a strong emphasis on low power consumption, caters to high-volume applications with stringent cost

considerations and space limitations, making them an ideal choice for a wide range of industries.

What sets the HTU21D(F) sensors apart is the meticulous attention to detail. Each sensor undergoes a rigorous calibration and testing process, ensuring that it meets the highest standards of performance. Lot identification information is prominently displayed on the sensor, providing clarity and traceability, while the chip itself stores an electronic identification code that can be accessed through a command interface. The sensors even possess the capability to detect low battery levels, thereby bolstering the overall reliability of their operation.

To further enhance their versatility, the HTU21D(F) sensors feature customizable resolution settings that can be adjusted through a straightforward command interface. This unique flexibility allows users to fine-tune their sensors to meet specific requirements, offering a range of resolution options, from 8/12bit to a remarkable 12/14bit for both humidity and temperature measurements.

## TEMPERATURE COEFFICIENT COMPENSATION EQUATION

Using the following temperature coefficient compensation equation will guarantee Relative Humidity accuracy given p.3, from 0°C to 80°C:

$$RH_{\text{compensated } T} = RH_{\text{actual } T} + (25 - T_{\text{actual}}) \times \text{CoeffTemp}$$

- $RH_{\text{actual } T}$  Ambient humidity in %RH, computed from HTU21D(F) sensor
- $T_{\text{actual}}$  Humidity cell temperature in °C, computed from HTU21D(F) sensor
- $\text{CoeffTemp}$  Temperature coefficient of the HTU21D(F) in %RH/°C

## Temperature

The HTU21D(F) sensor includes a temperature measurement feature as part of its capabilities. This digital humidity sensor is equipped to provide accurate temperature readings in addition to humidity data. The temperature measurement feature is integrated into the sensor's functionality, and the sensor delivers temperature readings in a digital format, making it easily accessible for microcontroller-based applications. Users can command the sensor to provide temperature data with adjustable resolution, ranging from 8-bit to 14-bit precision, to suit specific measurement requirements. This makes the HTU21D(F) sensor a versatile and comprehensive solution for applications that demand both humidity and temperature measurements with high accuracy and reliability.

## 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 <sup>(1)</sup>	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 <sup>(2)</sup>		T <sub>T</sub>		10		s

(1) Typical values are recommended for calculating energy consumption while maximum values shall be applied for calculating waiting times in communication.

(2) At 1m/s air flow.

- **Communication Protocol with HTU21D(F) Sensor: Initialization and Start Sequence**

### Powering Up the Sensor

To power the HTU21D(F) sensor, a voltage supply in the range of 1.5V to 3.6V is necessary. Upon power-up, the device undergoes an initialization phase, taking up to 15ms to reach its idle state (commonly known as sleep mode). During this period, the sensor is not ready to accept commands from the microcontroller. It's imperative not to send any commands before this initialization time elapses.

### Initiating the Start Sequence (S)

The communication process with the HTU21D(F) sensor commences with the issuance of a start bit. This start bit is generated by lowering the DATA line while keeping the SCK (Serial Clock) line high, followed by lowering the SCK line. This specific sequence signals the sensor that data transmission is about to begin.

- **Concluding the Communication Protocol: Stop Sequence (P)**

### **Implementing the Stop Sequence (P)**

To terminate data transmission with the HTU21D(F) sensor, a stop bit, or stop sequence, must be issued. This stop sequence is characterized by a change in the DATA line and the SCK line. Specifically, the DATA line is raised while the SCK line remains high, followed by raising the SCK line.

This stop sequence serves to signal the sensor that the current data transmission cycle is complete and prepares it for subsequent commands or interactions. It is a vital part of the communication protocol, ensuring the sensor's response and readiness for further communication or data retrieval. Properly adhering to both the start and stop sequences is essential for maintaining a reliable and efficient interaction with the HTU21D(F) sensor, ultimately leading to accurate and trustworthy data acquisition.

- **HTU21D(F) SENSOR LIST OF COMMANDS AND REGISTER ADDRESSES**

### **Sending a command**

After sending the start condition, the subsequent I<sup>2</sup>C header consist of a 7-bit I<sup>2</sup>C device address 0x40 and a DATA direction bit ('0' for Write access : 0x80). The HTU21D(F) sensor indicates the proper reception of a byte by pulling the DATA pin low (ACK bit) after the falling edge of the 8th SCK clock. After the issue of a measurement command (0xE3 for temperature, 0xE5 for relative humidity), the MCU must wait for the measurement to complete. The basic commands are given in the table below:

<b>Command</b>	<b>Code</b>	<b>Comment</b>
Trigger Temperature Measurement	0xE3	Hold master
Trigger Humidity Measurement	0xE5	Hold master
Trigger Temperature Measurement	0xF3	No Hold master
Trigger Humidity Measurement	0xF5	No Hold master
Write user register	0xE6	
Read user register	0xE7	
Soft Reset	0xFE	

Hold/No Hold master modes : There are two different operation modes to communicate with the HTU21D(F) sensor: Hold Master mode and No Hold Master mode.

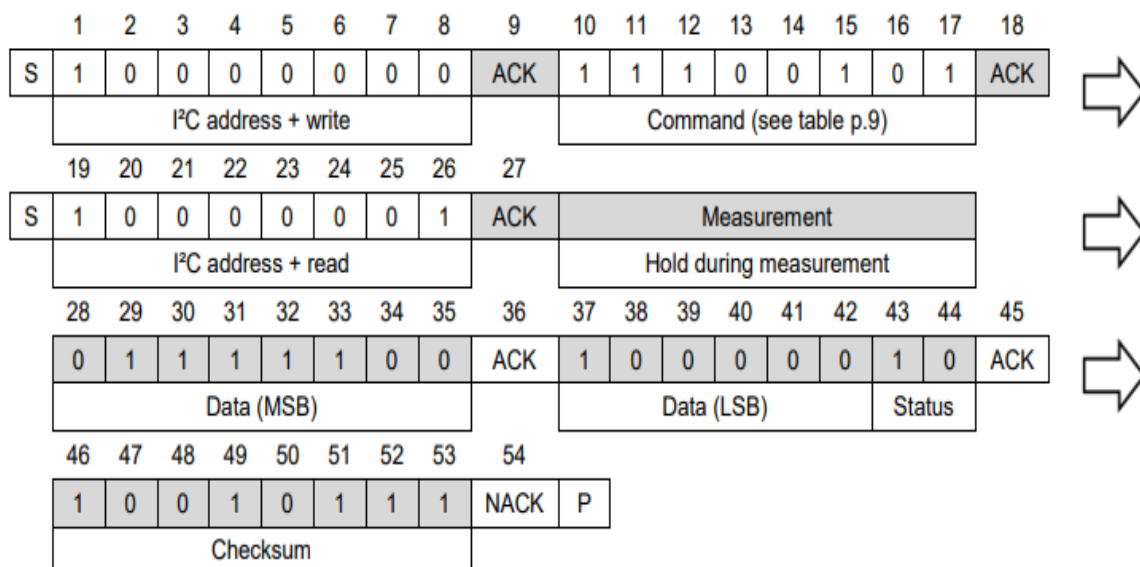
In the first case, the SCK line is blocked (controlled by HTU21D(F) sensor) during measurement process while in the second case the SCK line remain open for other communication while the sensor is processing the measurement.

No Hold Master mode allows for processing other I<sup>2</sup>C communication tasks on a bus while the HTU21D(F) sensor is measuring. A communication sequence of the two modes is available below.

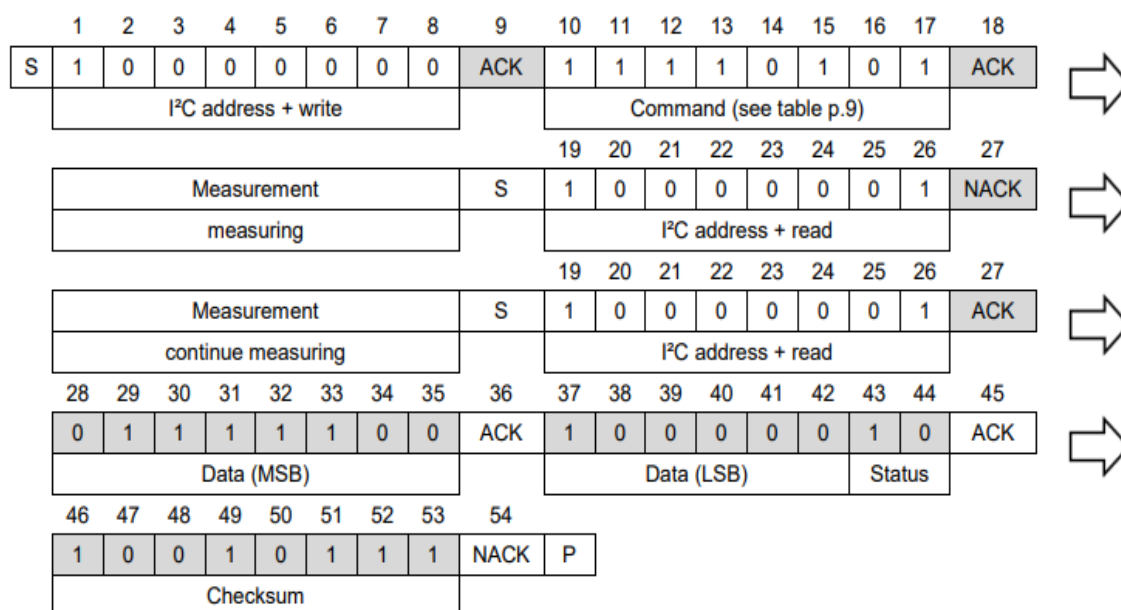
In the Hold Master mode, the HTU21D(F) pulls down the SCK line while measuring to force the master into a wait state. By releasing the SCK line, the HTU21D(F) sensor indicates that internal processing is completed and that transmission may be continued.

In the No Hold Master mode, the MCU has to poll for the termination of the internal processing of the HTU21D(F) sensor. This is done by sending a start condition followed by the I<sup>2</sup>C header ('1' for Read access: 0x81) as shown below. If the internal processing is finished, the HTU21D(F) sensor acknowledges the poll of the MCU and data can be read by the MCU. If the measurement processing is not finished, the HTU21D(F) sensor answers no ACK bit and start condition must be issued once more.

For both modes, since the maximum resolution of the measurement is 14 bits, the two last least significant bits (LSBs, bits 43 and 44) are used for transmitting status information. Bit 1 of the two LSBs indicates the measurement type ('0': temperature, '1': humidity). Bit 0 is currently not assigned.



*Hold Master communication sequence*



*No Hold Master communication sequence*

## RELATIVE HUMIDITY

(@T = 25°C, @Vdd = 3V)

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 Accuracy @25°C (20%RH to 80%RH)	typ			±2		%RH
	max		See graph 1			%RH
Replacement		fully interchangeable				
Temperature coefficient (from 0°C to 80°C)		T <sub>cc</sub>			-0.15	%RH/°C
Humidity Hysteresis				±1		%RH
Measuring Time <sup>(1)</sup>	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 <sup>(2)</sup>		T <sub>RH</sub>		5	10	s

<sup>(1)</sup> Typical values are recommended for calculating energy consumption while maximum values shall be applied for calculating waiting times in communication.

<sup>(2)</sup> At 1m/s air flow



```

/* 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 */

/* USER CODE END Includes */

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

/* USER CODE END PTD */

/* Private define -----
---*/
/* USER CODE BEGIN PD */
#define HTU21D_I2C_ADDRESS 64

/* 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 */

/* 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 */

    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        int temperature, humidity;
        uint8_t buffer[2];

```

```

        uint8_t command=0xE3;
        HAL_Delay(1000);
        HAL_I2C_Master_Transmit(&hi2c1, HTU21D_I2C_ADDRESS<<1, &command,
1, 500);
        HAL_I2C_Master_Receive(&hi2c1, HTU21D_I2C_ADDRESS<<1, buffer, 2,
500);
        uint16_t rawtemp=(buffer[0]<<8)|(buffer[1]);
        humidity=-6+(125.0*rawtemp/65536);
        temperature= -46.85+(175.72*rawtemp/65536);
        send(temperature,humidity);

        /* 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.HSISState = 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)
{
    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 send(int temperature, int humidity) {
    char buffer[100];
    sprintf(buffer, "Temperature: %d°C\n Humidity: %d\n",
temperature, humidity);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 500);
}

/* USER CODE END 4 */

/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */

```

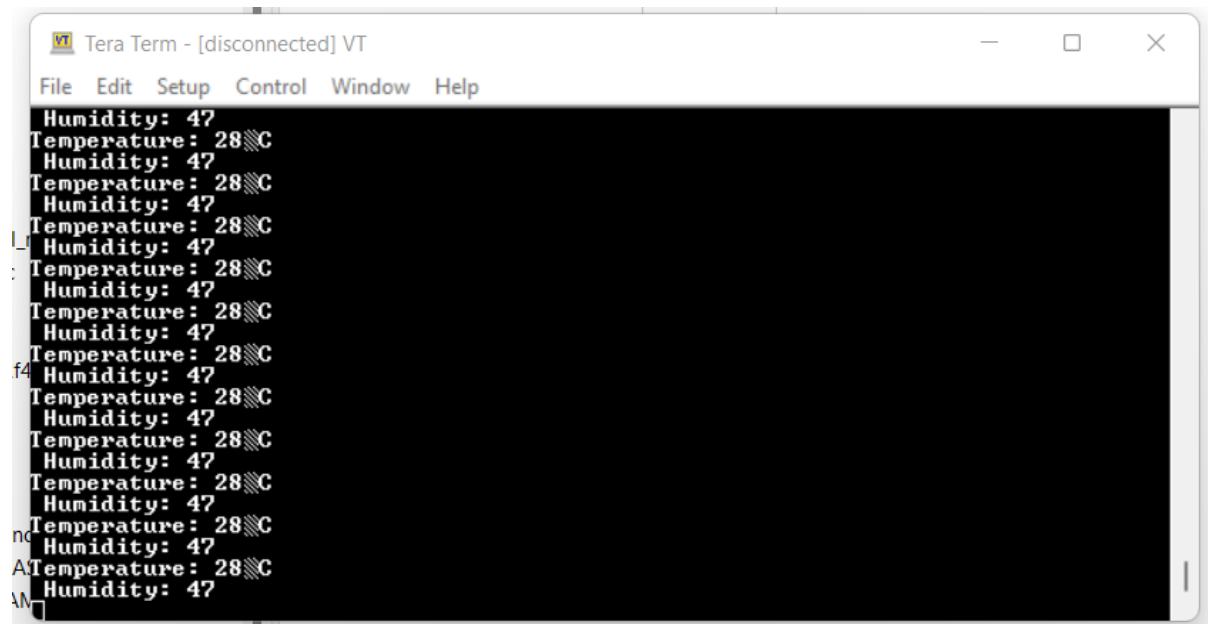
```

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 */

```

## OUTPUT:



The screenshot shows a Tera Term terminal window titled "Tera Term - [disconnected] VT". The window has a menu bar with "File", "Edit", "Setup", "Control", "Window", and "Help". The terminal output displays a repeating sequence of sensor data: "Humidity: 47" followed by "Temperature: 28°C". This sequence is repeated multiple times, with some lines partially obscured by the terminal's scrollback buffer. The output is displayed on a black background with white text.