**INTERFACING HTU21D WITH STM32F401RE**

**MINI PROJECT REPORT**

**DESHILA TECHNOLOGY RESEARCH INSTITUTE**

**MYSORE – 570006**

**&**

**SKANRAY TECHNOLOGIES PVT. LTD.**

**MYSORE – 570016**

**By,**

**PUNITH J**

**20068166**

**October 2023**

**Introduction:**

The HTU21D is a digital relative humidity and temperature sensor that has gained recognition for its accuracy, precision, and versatility in environmental monitoring applications. Manufactured by Sensirion, a leading sensor technology company, the HTU21D sensor offers high-performance capabilities for measuring humidity and temperature, making it a popular choice among engineers, researchers, and hobbyists.

With a reputation for its reliability and ease of integration, the HTU21D sensor is commonly used in various domains, including weather stations, indoor climate control systems, IoT (Internet of Things) devices, and environmental monitoring applications. Its digital output and compatibility with common microcontroller platforms, such as Arduino and Raspberry Pi, have contributed to its widespread adoption in projects involving data acquisition and environmental sensing.

The HTU21D sensor utilizes a combination of capacitive humidity sensing technology and a platinum resistance temperature sensor to provide accurate and precise measurements. Its compact form factor, low power consumption, and broad operational temperature range make it a valuable component in applications where humidity and temperature data are crucial for decision-making and control.

This introductory overview highlights the significance of the HTU21D sensor in the realm of environmental monitoring and sensing, setting the stage for a deeper exploration of its features, working principles, applications, and interfacing methods.

**KEY FEATURES OF THE HTU21D(F) SENSOR** :

1. **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.
2. **Temperature Sensing:** The HTU21D(F) can measure temperature in the range of -40°C to 125°C. It offers high accuracy in temperature measurements.
3. **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.
4. **Low Power Consumption:** The sensor is designed to operate with low power consumption, which is important for battery-powered and energy-efficient applications.
5. **Calibrated Sensor:** The HTU21D(F) is pre-calibrated, which means it can be started without the need for extensive calibration procedures.
6. **Compact Size:** The sensor is typically small and compact, making it suitable for applications with space constraints.
7. **Durable Design:** It is often built with protective features to make it robust and reliable in various environmental conditions.
8. **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.



HTU21D(F) Sensor

**PIN CONFIGURATION:**

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

1. **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.
2. **GND (Ground):** This is the ground connection for the sensor. It should be connected to the ground (0V) reference of your power supply.
3. **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.
4. **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.

**SENSOR SPECIFICATION:**

**Measurement Range:**

* Humidity: 0% to 100% RH (Relative Humidity).
* Temperature: -40°C to 125°C.

**Resolution:**

* Humidity: 0.04% RH.
* Temperature: 0.01°C.

**Accuracy:**

* Humidity: ±2% RH.
* Temperature: ±0.3°C.

**INTERFACING AND READING THE TEMPERATURE AND HUMIDITY:**

**Hardware Connections:**

**a. Power Supply:**

1. Connect the VDD pin of the HTU21D(F) sensor to a 3.3V power supply.
2. Connect the GND pin of the HTU21D(F) sensor to the ground (0V) of the power supply.
3. Connect the VDD pin of the STM32F401RETX to a 3.3V power supply.
4. Connect the GND pin of the STM32F401RETX to the ground (0V) of the power supply.

**b. I2C Interface:**

1. Connect the SDA (Serial Data) pin of the HTU21D(F) sensor to the SDA (I2C data) pin of the STM32F401RETX.
2. Connect the SCL (Serial Clock) pin of the HTU21D(F) sensor to the SCL (I2C clock) pin of the STM32F401RETX.

**Software Configuration (Using STM32CubeIDE):**

* Create a new STM32CubeIDE project for your STM32F401RETX microcontroller.
* Configure the project settings as needed, including selecting the target STM32 microcontroller.
* In STM32CubeIDE, open the "Pinout & Configuration" tab.
* Enable the I2C interface that you plan to use (I2C1, I2C2, etc.). Configure the pins as SDA and SCL.
* Set the appropriate I2C peripheral frequency.
* Save the Pinout configuration.
* In STM32CubeIDE, go to the "Peripherals" tab.
* Click on "I2C" and configure the I2C settings as needed (e.g., clock speed, addressing).
* Save the configuration.
* 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.
* Write code to read humidity and temperature data from the HTU21D sensor using I2C communication.
* Use a UART (serial communication) interface to transmit the data to the serial monitor.
* Compile your code in STM32CubeIDE.
* Flash the compiled code onto your STM32F401RETX microcontroller.
* Power up your hardware setup.
* Connect the microcontroller to a computer running a serial terminal Tera Term to view the temperature and humidity data transmitted via UART.

**Code:**

/\* 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**) {

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,

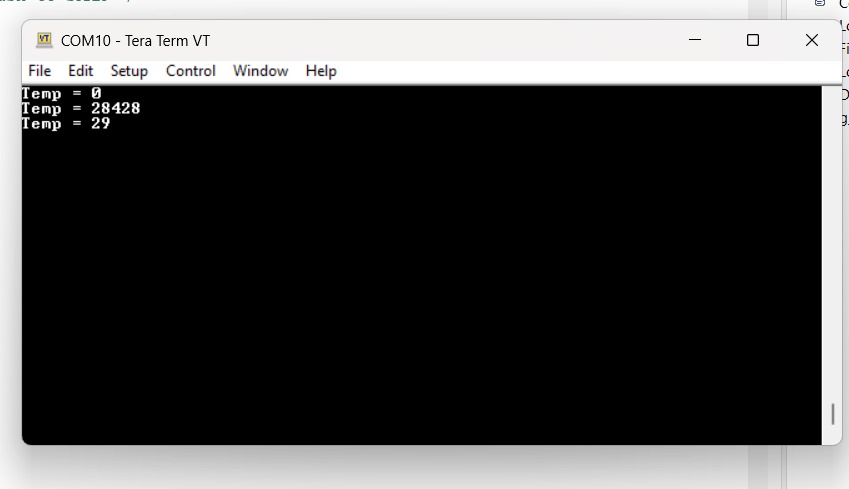
ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

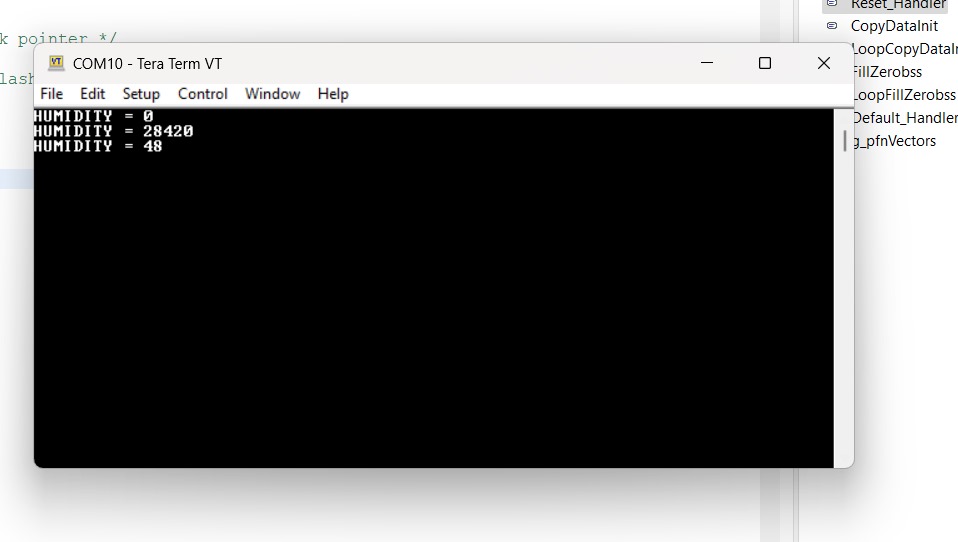
}

**#endif** /\* USE\_FULL\_ASSERT \*/

**Output:**



Temperature Reading in Serial Monitor



Humidity Reading in Serial Monitor