**Interfacing HTUD21 sensor with STM32F401RE to measure Temperature and Humidity of the vicinity.**

* Introduction:

**HTUD21 Sensor:**

The HTUD21 sensor is an advanced environmental sensor designed to monitor and provide accurate data on temperature and humidity conditions in various applications. Equipped with high-precision sensing elements, the HTUD21 sensor offers reliable measurements, making it ideal for use in climate control systems, weather monitoring, industrial processes, and IoT applications.

Its compact design, low power consumption, and digital output make it easy to integrate into a wide range of devices and systems, ensuring that users have access to essential environmental data for improved decision-making and environmental control.

* Sensor Specifications:

**Measuring range:**

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

**Accuracy:**

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

**Resolution:**

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

**Digital Interface**:

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

**Operating Voltage:**

The HTU21D sensor typically operates at 3.3V, making it suitable for use with microcontrollers and systems operating at 3.3V logic levels.

**Power Consumption:**

The sensor is designed to operate with low power consumption, making it suitable for battery-powered or energy-efficient devices.

**Calibration & Compact size:**

The sensor is pre-calibrated, which reduces the need for user calibration. And sensor is typically small and compact, making it suitable for applications with space constraints.

* Pin Configuration:

The sensor has relatively simple pin configuration. And it got four operating pins: VDD, GND, SDA, and SCL.



Fig: HTU21D Sensor

1. **VDD (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.
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.

* Interfacing with STM32F401RE Nucleo Board:

**Hardware** **Connections:**

The HTU21D sensor was connected to the STM32 Nucleo board as follows:

VCC: The VCC pin of the HTU21D sensor was connected to a 3.3V power supply on the Nucleo board.

GND: The GND pin of the HTU21D sensor was connected to the ground (GND) on the Nucleo board.

SDA: The SDA (data) pin of the HTU21D sensor was connected to the I2C data pin on the Nucleo board.

SCL: The SCL (clock) pin of the HTU21D sensor was connected to the I2C clock pin on the Nucleo board.

**Software Development:**

STM32CubeIDE was used for code development. This integrated development environment facilitates microcontroller configuration and code generation.

**Software Configuration:**

* Create a new STM32CubeIDE project for your STM32F401RETX microcontroller.
* In STM32CubeIDE, open the "Pinout & Configuration" tab.
* Enable the I2C interface that you plan to use (I2C1, I2C2, etc.). Configure the pins as I2C\_SDA and I2C\_SCL.
* Set the I2C peripheral frequency.
* Save the Pinout configuration.
* In STM32CubeIDE, go to the "Peripherals" tab.
* Click on "I2C" and configure the I2C settings as per the need.
* Save the configuration and generate the code.
* Make the changes in the code as per the concerns.
* Compile your code in STM32CubeIDE.
* Flash the compiled code onto your STM32F401RETX microcontroller.
* Additional UART can be enabled to display the real time data using TERA TERM application.
* 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**<stdio.h>

**#include**<string.h>

**#include**<stdint.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 \*/

/\* 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 \*/

**char** str[30]="";

uint8\_t cmd1 = 0xE3;

**char** buff[50]="";

**float** temp,hum;

HAL\_I2C\_Master\_Transmit(&hi2c1, 64<<1, &cmd1, 1, 1000);

HAL\_I2C\_Master\_Receive(&hi2c1, 64<<1, str, 2, 1000);

uint16\_t result=(str[0] << 8) | (str[1]);

temp= -46.85 + 175.72 \* ((**float**) result / 65536.0);

// hum= -6 + 125 \* ((**float**) result / 65535.0);

**memset**(buff ,0, **strlen**(buff));

**sprintf**(buff, "temp:%.2f C\n", temp);

HAL\_UART\_Transmit(&huart2, (uint8\_t\*)buff, **strlen**(buff), 1000);

/\* 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 \*/

/\* 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 \*/