**REAL-TIME TEMPERATURE AND HUMIDITY MONITORING USING HTU21D(F) SENSOR WITH STM32F401RE**

**INTRODUCTION**

The HTU21D(F) sensor represents a sophisticated digital relative humidity and temperature sensor designed for precision measurements in various environmental conditions. This sensor is commonly employed in applications that demand accurate monitoring of temperature and relative humidity.One of the primary features of the HTU21D(F) sensor is its capability to precisely measure relative humidity across a wide range, spanning from 0% to 100%. In addition to humidity, it is also adept at providing accurate temperature measurements within a range of -40°C to 125°C, making it a versatile solution for applications that necessitate comprehensive environmental data analysis.

The sensor communicates using a digital interface, commonly employing the I2C (Inter-Integrated Circuit) protocol, allowing seamless integration with microcontrollers and digital systems. Often factory-calibrated for accuracy, this sensor minimizes the need for user calibration, ensuring reliable and consistent data outputs.Compact in size and low in power consumption, the HTU21D(F) sensor is both efficient and adaptable for integration into a wide array of systems and devices. Its precision in providing temperature and humidity measurements, coupled with its digital compatibility, positions it as a valuable tool for applications that require real-time environmental monitoring and data-driven decision-making.

**OBJECTIVE**

1. Interface an HTU21D(F) sensor with an STM32F401RE microcontroller via the I2C protocol.

2. Retrieve accurate temperature and humidity data from the sensor.

3. Demonstrate the conversion and display of this data in readable formats for practical environmental monitoring applications.

**SENSORE SPECIFICATIONS**

Maximum Ratings:

Storage Temperature(Tstg) : -40 to 125°C

Supply Voltage (Peak-Vcc) : 3.8V

Humidity Operating Range(RH):0-100%RH

Temperature operating range(Ta):-40 to +125°C

Input current on any pin: -10 to +10 mA

Resolution:

a.For relative humidity

1)12bits – 0.04%RH 2)8bits – 0.7%RH

b.For Temperature

1)14bits – 0.01°C 2)12bits-0.04°C

Accuracy:

1. Relative Humidity Accuracy @25°C (20%RH to 80%RH) - ±2% RH
2. Temperature Accuracy @25°C - ±0.3°C

Response Time:

1. Relative Humidity(@33 to 75%RH) – 5S
2. Temperature(@15°C to 45°C) – 10S

**INTERFACE SPECIFICATION**

|  |  |  |
| --- | --- | --- |
| Number | Function | Comment |
| 1 | DATA | Data bit-stream |
| 2 | GND | Ground |
| 3 | NC | Must be left unconnected |
| 4 | NC | Must be left unconnected |
| 5 | VDD | Supply voltage |
| 6 | SCK | Selector for RH or Temp |
| PAD |  | Ground or unconnected |

**Power Pins (VDD, GND)**

The supply voltage of HTU21D(F) sensors must be in the range of 1.5VDC - 3.6VDC. Recommended supply voltage is 3VDC. However the typical application circuit includes a pull-up resistor R on data wire and a 100nF decoupling capacitor between VDD and GND, placed as close as possible to the sensor

**Serial clock input (SCK)**

SCK is used to synchronize the communication between microcontroller and HTU21D(F) sensor. Since the interface consists of fully static logic there is no minimum SCK frequency.

**Serial data (DATA)**

The DATA pin is used to transfer data in and out of the device. For sending a command to the HTU21D(F) sensor, DATA is valid on the rising edge of SCK and must remain stable while SCK is high. After the falling edge of SCK, the DATA value may be changed.

An external pull-up resistor (e.g. 10kΩ) on SCK is required to pull the signal high only for open collector or open drain technology microcontrollers

**HARDWARE** **SETUP:**

**Components** **Used:**

* STM32 Nucleo STM32F410 development board
* HTU21D digital humidity and temperature sensor
* Jumper wires

**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 SETUP:**

**Development Environment:**

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

**STM32CubeIDE:**

STM32CubeIDE is an Integrated Development Environment (IDE) specifically tailored for the STM32 family of microcontrollers by STMicroelectronics. It provides a comprehensive and user-friendly platform that combines the features of the Eclipse-based IDE with STM32CubeMX, a graphical tool for configuring STM32 microcontroller peripherals and generating initialization code. STM32CubeIDE offers a powerful set of tools including an editor, compiler, debugger, and various pre-configured software components that simplify and accelerate embedded software development for STM32 devices. With its intuitive interface, extensive device support, and numerous libraries, STM32CubeIDE streamlines the process of coding, debugging, and deploying applications for STM32 microcontrollers, catering to both seasoned developers and those new to embedded programming.

**Configuration in STM32CubeIDE:**

The following steps were taken in STM32CubeMX to configure the Nucleo board for interfacing with the HTU21D sensor:

* A new project was created for the STM32F410 Nucleo board.
* The I2C communication peripheral was enabled, and the pins for SDA and SCL were properly configured.
* Clock settings for the I2C peripheral were adjusted to match the sensor's requirements.

**Implementation:**

The code was developed to read data from the HTU21D sensor using the I2C interface. Additionally, a UART (USART2) was configured to communicate with Tera Term for displaying the data. The code is as follows,

/\* 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** <stdlib.h>

**#include** <string.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 \*/

//volatile char str[20] = "";

/\* 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**(**float** temp, **float** hum);

/\* 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 \*/

**unsigned** **char** command;

uint8\_t data[2] = { 0 }, flag = 1;

uint16\_t val;

**float** temp, hum;

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

**while** (1) {

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

**if** (flag) {

command = 0xE3;

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

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

val = ((data[0] << 8) | data[1]);

temp = -46.85 + 175.72 \* val / 65536;

flag = 0;

}

**if** (!flag) {

command = 0xE5;

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

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

val = ((data[0] << 8) | data[1]);

hum = -6 + ((125 \* val) / 65536);

flag = 1;

}

print(temp, hum);

}

/\* 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**(**float** temp, **float** hum) {

**char** str[60] = "";

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

**sprintf**(str, "temperature is %.2f and humidity is %.2f\n", temp, hum);

HAL\_UART\_Transmit(&huart2,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 \*/

**TERA TERM CONFIGURATION:**

a. Open Tera Term and select the appropriate serial port that corresponds to the USB-to-serial converter or the direct UART connection.

b. Configure the port settings (baud rate, data bits, stop bits, parity) to match the settings configured in STM32 code.

c. Click 'OK' to establish the connection.

**OBSERVATION:**

Upon successful programming of the STM32 Nucleo board, the HTU21D sensor was utilized to retrieve real-time temperature and humidity data, which was subsequently displayed in Tera Term via the established UART connection.

**CONCLUSION:**

In this project, successfully programmed the STM32 Nucleo board to collect real-time temperature and humidity data using the HTU21D sensor. By establishing an UART connection, able to display this data in Tera Term, providing a simple and efficient way to monitor the environment. This project showcases the seamless integration of hardware and software, enabling the board to accurately gather and transmit environmental data for various monitoring applications.