**DTRI Training-Mini Project 1**

**HTU21D- Digital Relative Humidity sensor with Temperature output**

**-Lavanya Srinuvasan**

The HTU21D is a digital humidity and temperature sensor. It is commonly used in various applications where accurate humidity and temperature measurements are required. HTU21D sensors are low power consumption sensors created for high-volume and cost-sensitive applications with extreme space constraints. The resolution of HTU21D sensor can be altered by command (8/12bit up to 12/14bit for RH/T), low battery can be detected and a checksum improves communication reliability.

**Key features**

* Uses the I2C interface
* Typical humidity accuracy of ±2%
* Typical temperature accuracy of ±0.3C
* Operates from 0 to 100% humidity but this sensor isn’t recommended for harsh environments where it could come in contact with water (such as rain).
* 3.3V sensor - use inline [logic level converters](https://learn.sparkfun.com/search/results?term=level+converter) or 10k resistors to limit 5V signals
* Only one HTU21D sensor can reside on the I2C bus at a time

**Applications**

* Automotive
* Medical
* Home appliances
* Printers
* Engine and Vehicle
* General industry
* Humidifiers
* HVAC systems
* Security cameras
* Weather monitoring
* Consumer goods
* Medical

**PERFORMANCE SPECIFICATION**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ratings** | **Symbol** | | **Value** | **Unit** | |
| Storage Temperature | Tstg | | -40 to 125 | °C | |
| Supply Voltage (Peak) | Vcc | | 3.8V | Vdc | |
| Humidity Operating Range | RH | | 0 to 100 | %RH | |
| Temperature Operating Range | Ta | | -40 to +125 | °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 |

Table 1. Performance Specification of HTU21D(F) sensor

**Pinout**

**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 (regulated).

**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.

|  |  |  |
| --- | --- | --- |
| **NO** | **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 | |

Table 2. Functions of HTU21D(F) sensor’s pins



Fig 1. Schematic of Master-Slave Configuration of HTU21D(F) sensor

**COMMUNICATION PROTOCOL WITH HTU21D(F) SENSOR**

**Start-up sensor**

The HTU21D(F) sensor requires a voltage supply between 1.5V and 3.6V. After power up, the device needs at most 15ms while SCK is high for reaching idle state (sleep mode), i.e to be ready accepting commands from the MCU. No command should be sent before that time.

**Start sequence (S)**

To initiate transmission, a start bit has to be issued. It consists of a lowering of the DATA line while SCK is high followed by lowering SCK.

**Stop sequence (P)**

To stop transmission, a stop bit has to be issued. It consists of a heightening of the DATA line while SCK is high preceded by a heightening of the SCK.

**Sending a command**

After sending the start condition, the subsequent I²C header consist of a 7-bit I²C 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:

|  |  |  |
| --- | --- | --- |
| **Command** | **Code** | **Comment** |
| 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 | |

Table 3. Basic commands to communicate with HTU21D(F) sensor

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 I²C communication tasks on a bus while the HTU21D(F) sensor is measuring.

**Algorithm**

Step 1:Start the program

Step 2: Initialize the necessary peripherals

Step 3: Initialize temperature and humidity variables as integers

Step 4: Initialize the commands 0xE3 and 0xE5 to set the master for temperature and humidity measurement.

Step 5: Set the HAL\_I2C\_Master\_Transmit function which is used to send data over the I2C (Inter-Integrated Circuit) bus in master mode.

Step 6: Set the HAL\_I2C\_Master\_Receive function which is used for receiving data over the I2C (Inter-Integrated Circuit) bus in master mode.

Step 7: Calculate the humidity and temperature using the formulae from the specific sheet.

Step 8: View the output in tera term application by transmitting the sensor data 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 \*/

/\* 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 \*/

**int** temperature,humidity;

uint8\_t buffer[4];

uint8\_t command = 0xE3;

uint8\_t command = 0xE5;

HAL\_Delay(5);

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);

//command = 0xE5;

/\*//HAL\_I2C\_Master\_Transmit(&hi2c1, HTU21D\_I2C\_ADDRESS, &command, 1,

500);

HAL\_I2C\_Master\_Receive(&hi2c1, HTU21D\_I2C\_ADDRESS, buffer, 2,

500);

uint16\_t rawhumidity = (buffer[0] << 8) | (buffer[1] );

\*humidity = -6 + (125.0 \* rawhumidity / 65536.0);\*/

SendDataOverSerial(temperature,humidity);

/\* 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** **SendDataOverSerial**(**int** temperature,**int** humidity) {

**char** buffer[100] = " ";

memset(buffer, 0, strlen(buffer));

**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**

In the Tera Term application, the temperature and humidity data transferred from STM32F401RE microcontroller via UART communication protocol is displayed.

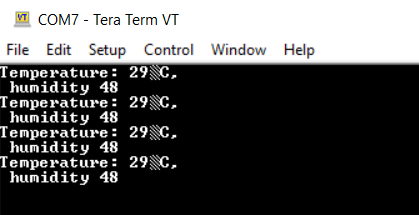
****

Fig 2. Temperature and humidity output from HTU21D(F) sensor