**Temperature sensor**

Temperature sensors are devices that detect and measure coldness and heat and convert it into an electrical signal. Temperature sensors are utilized in our daily lives, be it in the form of domestic water heaters, thermometers, refrigerators, or microwaves. There is a wide range of applications of temperature sensors, including the geotechnical monitoring field.

When not in use, the sensor can be put in power down mode.

Supported temperature range: –40 to 125 °C .

Precision: ±1.5 °C.

Select ADC1\_IN16 or ADC1\_IN18 input channel. 4. Select a sampling time greater than the minimum sampling time specified in the datasheet. 5. Set the TSVREFE bit in the ADC\_CCR register to wake up the temperature sensor from power down mode 6. Start the ADC conversion by setting the SWSTART bit (or by external trigger) 7. Read the resulting VSENSE data in the ADC data register 8. Calculate the temperature using the following formula: Temperature (in °C) = {(VSENSE – V25) / Avg\_Slope} + 25 Where: – V25 = VSENSE value for 25° C – Avg\_Slope = average slope of the temperature vs. VSENSE curve (given in mV/°C or µV/°C) Refer to the datasheet’s electrical characteristics section for the actual values of V25 and Avg\_Slope.

HAL\_I2C\_Master\_Transmit(I2C\_HandleTypeDef \*hi2c, uint16\_t DevAddress, uint8\_t \*pData, uint16\_t Size, uint32\_t Timeout)

\*hi2c - I2C structure  
DevAddress - Device address

\*pData - I don't understand this parameter.  
this pointer is used to transfer the register address in the Slave device.

Size - data size in Bytes.

**memset**(buff,0,**strlen**(buff)) : clears the array.

Sprint() : Used to print the formatted string.

HAL\_UART\_Transmit() : each call would lock that uart, send the data, then unlock the uart. If you ever got a receive interrupt while HAL\_UART\_Transmit() was active, the call to HAL\_UART\_Receive\_IT() in the interrupt callback would fail (and return) because the uart is locked by HAL\_UART\_Transmit(). So no more receive interrupts.

**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;

uint8\_t cmd2 = 0xE5;

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

**float** temp,hum;

HAL\_Delay(5);

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

HAL\_I2C\_Master\_Transmit(&hi2c1, 64<<1, &cmd2, 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 hum:%.2f\n",temp,hum);

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 \*/