GROUP7 STM32 ENVIRONSENSING HAT

1.0

Generated by Doxygen 1.9.6

Chapter 1

EEPROM Library Integration Guide

This guide provides instructions on how to integrate the EEPROM library into your STM32 project using STM32Cube IDE or any other compatible IDE.

1.1 Step 1: Download the library files

Download the following EEPROM library files:

1. `EEPROM.h` - The header file containing the function prototypes and necessary definitions.
2. `EEPROM.c` - The source file containing the function implementations.

1.2 Step 2: Add the library files to your project

Follow these steps to add the EEPROM library files to your project:

1.2.1 STM32Cube IDE

1. In STM32Cube IDE, open your STM32 project.
2. Navigate to the project tree in the "Project Explorer" tab.
3. Place the `EEPROM.h` file into the "Inc" folder (or the folder where header files are stored in your project).
4. Place the `EEPROM.c` file into the "Src" folder (or the folder where source files are stored in your project).

1.2.2 Other IDEs

1. Open your STM32 project in the IDE you are using.
2. Place the `EEPROM.h` file in the folder where header files are stored in your project (usually an "include" or "inc" folder).
3. Place the `EEPROM.c` file in the folder where source files are stored in your project (usually a "source" or "src" folder).

1.3 Step 3: Include the library header in main.c

In the main.c file of your project, add the following include statement at the beginning of the file, along with other include statements:

```
#include "EEPROM.h"
```

1.4 Step 4: Usage Examples

ere are some examples of how to use the EEPROM library in your STM32 project:

1.4.1 Write data to EEPROM

```
 uint8_t data_to_write[] = "Hello, EEPROM!";
EEPROM_Write(0, 0, data_to_write, sizeof(data_to_write));
```

1.4.2 Read data from EEPROM

```
uint8_t read_buffer[16];
EEPROM_Read(0, 0, read_buffer, sizeof(read_buffer));
```

1.4.3 Erase a page in EEPROM

```
EEPROM_PageErase(2);
```

1.4.4 Comprenhensive example on how to use it in main.c

```
#include "main.h"
#include "EEPROM.h"

I2C_HandleTypeDef hi2c1;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);

int main(void)
{
    HAL_Init();
    SystemClock_Config();

    MX_GPIO_Init();
    MX_I2C1_Init();

    // Write data to EEPROM
    uint8_t data_to_write[] = "Hello, EEPROM!";
    EEPROM_Write(0, 0, data_to_write, sizeof(data_to_write));

    // Read data from EEPROM
    uint8_t read_buffer[16];
    EEPROM_Read(0, 0, read_buffer, sizeof(read_buffer));

    // Write a float number to EEPROM
    float number_to_write = 3.14159265;
    EEPROM_Write_NUM(1, 0, number_to_write);

    // Read a float number from EEPROM
    float read_number;
    read_number = EEPROM_Read_NUM(1, 0);

    // Erase a page in EEPROM
    EEPROM_PageErase(2);

    while (1)
    {
        // Main loop
    }
```

```
}

void SystemClock_Config(void)
{
    // System clock configuration code...
}

static void MX_GPIO_Init(void)
{
    // GPIO initialization code...
}

static void MX_I2C1_Init(void)
{
    // I2C1 initialization code...
}
```

Chapter 2

LDR Library Integration Guide

This guide provides instructions on how to integrate the LDR library into your STM32 project using STM32Cube IDE or any other compatible IDE.

2.1 Step 1: Download the library files

Download the following LDR library files:

1. `ldr.h` - The header file containing the function prototypes and necessary definitions.
2. `ldr.c` - The source file containing the function implementations.

2.2 Step 2: Add the library files to your project

Follow these steps to add the LDR library files to your project:

2.2.1 STM32Cube IDE

1. In STM32Cube IDE, open your STM32 project.
2. Navigate to the project tree in the "Project Explorer" tab.
3. Place the `ldr.h` file into the "Inc" folder (or the folder where header files are stored in your project).
4. Place the `ldr.c` file into the "Src" folder (or the folder where source files are stored in your project).

2.2.2 Other IDEs

1. Open your STM32 project in the IDE you are using.
2. Place the `ldr.h` file in the folder where header files are stored in your project (usually an "include" or "inc" folder).
3. Place the `ldr.c` file in the folder where source files are stored in your project (usually a "source" or "src" folder).

2.3 Step 3: Include the library header in main.c

In the `main.c` file of your project, add the following include statement at the beginning of the file, along with other include statements:

```
#include "ldr.h"
```

2.4 Step 4: Initialize the LDR in main.c(sets the calibration constants)

In the`main.c` function of your project, call the `LDR_Init()` function to initialize the LDR:

```
int main(void)
{
    // Initialize peripherals, system clock, etc.

    // Initialize LDR
    LDR_Init();

    // Other code
}
```

2.5 Step 5: Read analog light intensity in main.c

To read the analog light intensity using the LDR, call the `LDR_ReadAnalogLightIntensity()` function in your code:

```
int main(void)
{
    // Initialize peripherals, system clock, etc.

    // Initialize LDR
    LDR_Init();

    // Read analog light intensity
    float light_intensity = LDR_ReadAnalogLightIntensity(&hadc);

    // Other code
}
```

Again, `hadc` is a handle to the ADC peripheral that you should have already initialized in your project.

Chapter 3

LTR-303ALS Ambient Light Sensor Integration Guide

This guide will help you integrate the `ltr303als.h` and `ltr303als.c` files into your STM32 project using the LTR-303ALS Ambient Light Sensor.

3.1 Step 1: Download the library files

Download the following files:

- `ltr303als.h`
- `ltr303als.c`

3.2 Step 2: Add the library files to your project

1. Open your STM32 project in the STM32CubeIDE or your preferred IDE.
2. Copy the `ltr303als.h` file into the `Inc` folder of your project.
3. Copy the `ltr303als.c` file into the `Src` folder of your project.

3.3 Step 3: Include the library in your code

In the source file where you want to use the LTR-303ALS library (usually `main.c`), add the following line at the beginning of the file:

```
#include "ltr303als.h"
```

3.4 Step 4: Initialize the sensor

Initialize the LTR-303ALS sensor by calling the `LTR303ALS_Init` function, passing a pointer to an `I2C_HandleTypeDef` structure and the desired integration time and measurement rate.

```
I2C_HandleTypeDef hi2c1; // This should be configured and initialized using HAL

if (LTR303ALS_Init(&hi2c1, INTEGRATION_TIME, MEASUREMENT_RATE) != HAL_OK) {
    // Handle initialization error
}
```

3.5 Step 5: Read light intensity

To read the light intensity from the sensor, call the `LTR303ALS_ReadLightIntensity` function, passing a pointer to an `I2C_HandleTypeDef` structure and pointers to two `uint16_t` variables that will store the values of channels 0 and 1.

```
uint16_t ch0, ch1;

if (LTR303ALS_ReadLightIntensity(&hi2c1, &ch0, &ch1) != HAL_OK) {
    // Handle read error
} else {
    // Process light intensity data
}
```

Chapter 4

RTC Library Library Integration Guide

4.1 Step 1:

Download the `rtc.c` and `rtc.h` files from the provided source or from your own implementation.

4.2 Step 2:

Add the `rtc.c` file to your project's source folder and the `rtc.h` file to your project's include folder.

4.3 Step 3:

In your main program, add the following include statement to include the RTC library header file:

```
#include "rtc.h"
```

4.4 Step 4:

Initialize the RTC module using STM32CubeIDE GUI. Configure RTC clock source and enable RTC in the main initialization code or in the Clock Configuration function (SystemClock_Config).

4.5 Step 5:

In your code, call the `RTC_SetTime()` function to set the RTC time. The function takes the `RTC_Handle↩TypeDef` structure and three parameters for hours, minutes, and seconds. The function returns a `HAL_Status↩TypeDef` value indicating the success or failure of the operation.

```
uint8_t hours = 12;
uint8_t minutes = 30;
uint8_t seconds = 0;
HAL_StatusTypeDef status = RTC_SetTime(&hrtc, hours, minutes, seconds);
if (status != HAL_OK) {
    // Handle error
}
```

4.6 Step 6:

In your code, call the `RTC_GetTime()` function to get the RTC time. The function takes the `RTC_HandleTypeDef` structure and three pointers to `uint8_t` variables for hours, minutes, and seconds. The function returns a `HAL_StatusTypeDef` value indicating the success or failure of the operation.

```
uint8_t hours, minutes, seconds;
HAL_StatusTypeDef status = RTC_GetTime(&hrtc, &hours, &minutes, &seconds);
if (status != HAL_OK) {
    // Handle error
}
```

4.7 Step 7:

In your code, call the `RTC_SetDate()` function to set the RTC date.

```
uint8_t day = 11;
uint8_t month = 5;
uint8_t year = 23; // It's usually the year minus 2000.

RTC_SetDate(&hrtc, day, month, year);
```

Now, the RTC date is set to 11th of May, 2023.

4.8 Step 8:

In your code, call the `RTC_GetDate()` function to get the RTC date. You can retrieve the date using `RTC_GetDate` as follows:

```
uint8_t day;
uint8_t month;
uint8_t year;

RTC_GetDate(&hrtc, &day, &month, &year);

printf("Current date: %02d-%02d-%02d\n", day, month, year + 2000); // Adding 2000 to get the full year.
```

4.9 Step 9:

Use the values of the `hours`, `minutes`, `seconds`,`day` ,`month`,`year`variables as needed in your program.

Note: The RTC module needs an external battery to retain the time when the device is powered off. The battery must be connected to the Vbat pin of the STM32 microcontroller. An external battery is not used in our setup so the time needs to be reset every time the STM32 goes off. However an external battery can be implemented to overcome this.

4.10 Example:

Below is an example:

```
    #include "rtc.h"

int main(void) {
    RTC_HandleTypeDef hrtc;
    uint8_t hours = 12;
    uint8_t minutes = 30;
    uint8_t seconds = 0;
    HAL_StatusTypeDef status = RTC_SetTime(&hrtc, hours, minutes, seconds);
    if (status != HAL_OK) {
        // Handle error
```

```
    }

    uint8_t read_hours, read_minutes, read_seconds;
    status = RTC_GetTime(&hrtc, &read_hours, &read_minutes, &read_seconds);
    if (status != HAL_OK) {
        // Handle error
    }
   uint8_t day = 11;
   uint8_t month = 5;
   uint8_t year = 23;
   RTC_SetDate(&hrtc, day, month, year);
   uint8_t day;
   uint8_t month;
   uint8_t year;

   RTC_GetDate(&hrtc, &day, &month, &year);




    // Use the read hours, minutes and seconds values and the date values
    // ...

    while (1) {
        // Main loop
    }
}
```

Chapter 5

TMP102 Library Integration Guide

This guide provides instructions on how to integrate the TMP102 temperature sensor library into your STM32 project using STM32Cube IDE or any other compatible IDE.

5.1 Step 1: Download the library files

Download the following TMP102 library files:

1. `tmp102.h` - The header file containing the function prototypes and necessary definitions.
2. `tmp102.c` - The source file containing the function implementations.

5.2 Step 2: Add the library files to your project

Follow these steps to add the TMP102 library files to your project:

5.2.1 STM32Cube IDE

1. In STM32Cube IDE, open your STM32 project.
2. Navigate to the project tree in the "Project Explorer" tab.
3. Place the `tmp102.h` file into the "Inc" folder (or the folder where header files are stored in your project).
4. Place the `tmp102.c` file into the "Src" folder (or the folder where source files are stored in your project).

5.2.2 Other IDEs

1. Open your STM32 project in the IDE you are using.
2. Place the `tmp102.h` file in the folder where header files are stored in your project (usually an "include" or "inc" folder).
3. Place the `tmp102.c` file in the folder where source files are stored in your project (usually a "source" or "src" folder).

5.3 Step 3: Include the library header in main.c

In the `main.c` file of your project, add the following include statement at the beginning of the file, along with other include statements:

```
#include "tmp102.h"
```

5.4 Step 4: Initialize the I2C interface

Before using the TMP102 sensor, you need to initialize the I2C interface. This can be done using the `HAL_I2C_Init()` function provided by the STM32 HAL library. Here's an example:

```
I2C_HandleTypeDef hi2c1;

void SystemClock_Config(void);

int main(void) {
  // Initialize HAL and system clock
  HAL_Init();
  SystemClock_Config();

  // Initialize I2C1
  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x00707CBB;
  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();
  }
}
```

5.5 Step 5: Initialize the TMP102 sensor and read Temperature Value

After initializing the I2C interface, you can now initialize the TMP102 sensor. This can be done using the TMP102_Init() function provided by the library. Here's an example:

```
#include "tmp102.h"

I2C_HandleTypeDef hi2c1;

int main(void)
{
  HAL_Init();
  MX_I2C1_Init();

  // Initialize TMP102 sensor
  TMP102_Init(&hi2c1);//Use the I2C handle intiliased
  while (1)
  {
    // Read temperature value
    float temperature = TMP102_ReadTemperature(&hi2c1);

    // Do something with temperature value
  }
}
```

Chapter 6

Module Index

6.1 Modules

Here is a list of all modules:

Chapter 7

File Index

7.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 8

Module Documentation

8.1 CMSIS

Modules

- Stm32f0xx_system

8.1.1 Detailed Description

8.2 Stm32f0xx_system

Modules

- STM32F0xx_System_Private_Includes
- STM32F0xx_System_Private_TypesDefinitions
- STM32F0xx_System_Private_Defines
- STM32F0xx_System_Private_Macros
- STM32F0xx_System_Private_Variables
- STM32F0xx_System_Private_FunctionPrototypes
- STM32F0xx_System_Private_Functions

8.2.1 Detailed Description

8.3 STM32F0xx_System_Private_Includes

8.4 STM32F0xx_System_Private_TypesDefinitions

8.5 STM32F0xx_System_Private_Defines

Macros

- #define HSE_VALUE ((uint32_t)8000000)
- #define HSI_VALUE ((uint32_t)8000000)
- #define HSI48_VALUE ((uint32_t)48000000)

8.5.1 Detailed Description

8.5.2 Macro Definition Documentation

8.5.2.1 HSE_VALUE

```
#define HSE_VALUE ((uint32_t)8000000)
```

Default value of the External oscillator in Hz. This value can be provided and adapted by the user application.

8.5.2.2 HSI48_VALUE

```
#define HSI48_VALUE ((uint32_t)48000000)
```

Default value of the HSI48 Internal oscillator in Hz. This value can be provided and adapted by the user application.

8.5.2.3 HSI_VALUE

```
#define HSI_VALUE ((uint32_t)8000000)
```

Default value of the Internal oscillator in Hz. This value can be provided and adapted by the user application.

8.6 STM32F0xx_System_Private_Macros

8.7 STM32F0xx_System_Private_Variables

Variables

- uint32_t **SystemCoreClock** = 8000000
- const uint8_t **AHBPrescTable** [16] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}
- const uint8_t **APBPrescTable** [8] = {0, 0, 0, 0, 1, 2, 3, 4}

8.7.1 Detailed Description

8.8 STM32F0xx_System_Private_FunctionPrototypes

8.9 STM32F0xx_System_Private_Functions

Functions

- void SystemInit (void)

  Setup the microcontroller system.
- void SystemCoreClockUpdate (void)

  Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

8.9.1 Detailed Description

8.9.2 Function Documentation

8.9.2.1 SystemCoreClockUpdate()

```
void SystemCoreClockUpdate (
            void  )
```

Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

Note

Each time the core clock (HCLK) changes, this function must be called to update SystemCoreClock variable value. Otherwise, any configuration based on this variable will be incorrect.

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:

- If SYSCLK source is HSI, SystemCoreClock will contain the HSI_VALUE(∗)
- If SYSCLK source is HSE, SystemCoreClock will contain the HSE_VALUE(∗∗)
- If SYSCLK source is PLL, SystemCoreClock will contain the HSE_VALUE(∗∗) or HSI_VALUE(∗) multiplied/divided by the PLL factors.
- If SYSCLK source is HSI48, SystemCoreClock will contain the HSI48_VALUE(∗∗∗)

(∗) HSI_VALUE is a constant defined in stm32f0xx_hal_conf.h file (default value 8 MHz) but the real value may vary depending on the variations in voltage and temperature.

(∗∗) HSE_VALUE is a constant defined in stm32f0xx_hal_conf.h file (its value depends on the application requirements), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.

(∗∗∗) HSI48_VALUE is a constant defined in stm32f0xx_hal_conf.h file (default value 48 MHz) but the real value may vary depending on the variations in voltage and temperature.

- The result of this function could be not correct when using fractional value for HSE crystal.

Parameters

None	

Return values

None	

8.9.2.2 SystemInit()

```
void SystemInit (
             void  )
```

Setup the microcontroller system.

Parameters

None	

Return values

None	

Chapter 9

File Documentation

9.1 AT24C256/EEPROM.c File Reference

Using the HAL I2C Functions.

```
#include "EEPROM.h"
#include "math.h"
#include "string.h"
```

Macros

- #define **EEPROM_I2C** &hi2c1
- #define **EEPROM_ADDR** 0xA0
- #define **PAGE_SIZE** 64
- #define **PAGE_NUM** 512

Functions

- void EEPROM_Write (uint16_t page, uint16_t offset, uint8_t *data, uint16_t size)

  Write data to the EEPROM.
- void EEPROM_Write_NUM (uint16_t page, uint16_t offset, float data)

  Write a float/integer value to the EEPROM.
- float EEPROM_Read_NUM (uint16_t page, uint16_t offset)

  Read a single float/integer value from the EEPROM.
- void EEPROM_Read (uint16_t page, uint16_t offset, uint8_t *data, uint16_t size)

  Read data from the EEPROM.
- void EEPROM_PageErase (uint16_t page)

  Erase a page in the EEPROM Memory.

Variables

- uint8_t **bytes_temp** [4]
- I2C_HandleTypeDef **hi2c1**

9.1.1 Detailed Description

Using the HAL I2C Functions.

Author

ControllersTech

Date

Feb 16, 2021

Attention

9.1.2 Function Documentation

9.1.2.1 EEPROM_PageErase()

```
void EEPROM_PageErase (
            uint16_t page )
```

Erase a page in the EEPROM Memory.

Parameters

page	Number of page to erase In order to erase multiple pages, just use this function in the for loop

Return values

None	

9.1.2.2 EEPROM_Read()

```
void EEPROM_Read (
            uint16_t page,
```

```
            uint16_t offset,
            uint8_t * data,
            uint16_t size )
```

Read data from the EEPROM.

Parameters

page	Number of the start page. Range from 0 to PAGE_NUM-1.
offset	Start byte offset in the page. Range from 0 to PAGE_SIZE-1.
data	Pointer to the data to write in bytes.
size	Size of the data.

Return values

None.	

9.1.2.3 EEPROM_Read_NUM()

```
float EEPROM_Read_NUM (
            uint16_t page,
            uint16_t offset )
```

Read a single float/integer value from the EEPROM.

Read a float or integer value from the EEPROM.

Parameters

page	Number of the start page. Range from 0 to PAGE_NUM-1.
offset	Start byte offset in the page. Range from 0 to PAGE_SIZE-1.

Return values

Float/integer	value.

9.1.2.4 EEPROM_Write()

```
void EEPROM_Write (
            uint16_t page,
            uint16_t offset,
            uint8_t * data,
            uint16_t size )
```

Write data to the EEPROM.

Parameters

page	Start page number (0 to PAGE_NUM-1).
offset	Start byte offset in the page (0 to PAGE_SIZE-1).
data	Pointer to the data to write in bytes.
size	Size of the data.

Return values

None	

9.1.2.5 EEPROM_Write_NUM()

```
void EEPROM_Write_NUM (
            uint16_t page,
            uint16_t offset,
            float data )
```

Write a float/integer value to the EEPROM.

Write a float or integer value to the EEPROM.

Parameters

page	Number of the start page. Range from 0 to PAGE_NUM-1.
offset	Start byte offset in the page. Range from 0 to PAGE_SIZE-1.
data	Float/integer value that you want to write.

Return values

None.	

9.2 SENSEHAT INTEGRATION/Core/Src/EEPROM.c File Reference

Using the HAL I2C Functions.

```
#include "EEPROM.h"
#include "math.h"
#include "string.h"
```

Macros

- #define **EEPROM_I2C** &hi2c1
- #define **EEPROM_ADDR** 0xA0
- #define **PAGE_SIZE** 64
- #define **PAGE_NUM** 512

Functions

- void EEPROM_Write (uint16_t page, uint16_t offset, uint8_t *data, uint16_t size)
  Write data to the EEPROM.
- void EEPROM_Write_NUM (uint16_t page, uint16_t offset, float data)
  Write a float/integer value to the EEPROM.
- float EEPROM_Read_NUM (uint16_t page, uint16_t offset)
  Read a single float/integer value from the EEPROM.
- void EEPROM_Read (uint16_t page, uint16_t offset, uint8_t *data, uint16_t size)
  Read data from the EEPROM.
- void EEPROM_PageErase (uint16_t page)
  Erase a page in the EEPROM Memory.

Variables

- uint8_t **bytes_temp** [4]
- I2C_HandleTypeDef **hi2c1**

9.2.1 Detailed Description

Using the HAL I2C Functions.

Author

ControllersTech

Date

Feb 16, 2021

Attention

9.2.2 Function Documentation

9.2.2.1 EEPROM_PageErase()

```
void EEPROM_PageErase (
            uint16_t page )
```

Erase a page in the EEPROM Memory.

Parameters

page	Number of page to erase In order to erase multiple pages, just use this function in the for loop

Return values

None	

9.2.2.2 EEPROM_Read()

```
void EEPROM_Read (
            uint16_t page,
            uint16_t offset,
            uint8_t * data,
            uint16_t size )
```

Read data from the EEPROM.

Parameters

page	Number of the start page. Range from 0 to PAGE_NUM-1.
offset	Start byte offset in the page. Range from 0 to PAGE_SIZE-1.
data	Pointer to the data to write in bytes.
size	Size of the data.

Return values

None.	

9.2.2.3 EEPROM_Read_NUM()

```
float EEPROM_Read_NUM (
            uint16_t page,
            uint16_t offset )
```

Read a single float/integer value from the EEPROM.

Read a float or integer value from the EEPROM.

Parameters

page	Number of the start page. Range from 0 to PAGE_NUM-1.
offset	Start byte offset in the page. Range from 0 to PAGE_SIZE-1.

Return values

Float/integer	value.

9.2.2.4 EEPROM_Write()

```
void EEPROM_Write (
            uint16_t page,
            uint16_t offset,
            uint8_t * data,
            uint16_t size )
```

Write data to the EEPROM.

Parameters

page	Start page number (0 to PAGE_NUM-1).
offset	Start byte offset in the page (0 to PAGE_SIZE-1).
data	Pointer to the data to write in bytes.
size	Size of the data.

Return values

None	

9.2.2.5 EEPROM_Write_NUM()

```
void EEPROM_Write_NUM (
            uint16_t page,
            uint16_t offset,
            float data )
```

Write a float/integer value to the EEPROM.

Write a float or integer value to the EEPROM.

Parameters

page	Number of the start page. Range from 0 to PAGE_NUM-1.
offset	Start byte offset in the page. Range from 0 to PAGE_SIZE-1.
data	Float/integer value that you want to write.

Return values

None.	

9.3 AT24C256/EEPROM.h File Reference

Using the HAL I2C Functions.

```
#include "stdint.h"
#include "stm32f0xx_hal.h"
```

Functions

- void EEPROM_Write (uint16_t page, uint16_t offset, uint8_t *data, uint16_t size)
  Write data to the EEPROM.
- void EEPROM_Read (uint16_t page, uint16_t offset, uint8_t *data, uint16_t size)
  Read data from the EEPROM.
- void EEPROM_PageErase (uint16_t page)
  Erase a page in the EEPROM Memory.
- void EEPROM_Write_NUM (uint16_t page, uint16_t offset, float fdata)
  Write a float or integer value to the EEPROM.
- float EEPROM_Read_NUM (uint16_t page, uint16_t offset)
  Read a float or integer value from the EEPROM.

9.3.1 Detailed Description

Using the HAL I2C Functions.

Author

ControllersTech

Date

Feb 16, 2021

Attention

9.3.2 Function Documentation

9.3.2.1 EEPROM_PageErase()

```
void EEPROM_PageErase (
            uint16_t page )
```

Erase a page in the EEPROM Memory.

Parameters

page	Page number to erase.

Return values

None	

Parameters

page	Number of page to erase In order to erase multiple pages, just use this function in the for loop

Return values

None	

9.3.2.2 EEPROM_Read()

```
void EEPROM_Read (
            uint16_t page,
            uint16_t offset,
            uint8_t * data,
            uint16_t size )
```

Read data from the EEPROM.

Parameters

page	Start page number (0 to PAGE_NUM-1).
offset	Start byte offset in the page (0 to PAGE_SIZE-1).
data	Pointer to the data to read in bytes.
size	Size of the data.

Return values

None	

Parameters

page	Number of the start page. Range from 0 to PAGE_NUM-1.
offset	Start byte offset in the page. Range from 0 to PAGE_SIZE-1.
data	Pointer to the data to write in bytes.
size	Size of the data.

Return values

None.	

9.3.2.3 EEPROM_Read_NUM()

```
float EEPROM_Read_NUM (
            uint16_t page,
            uint16_t offset )
```

Read a float or integer value from the EEPROM.

Parameters

page	Start page number (0 to PAGE_NUM-1).
offset	Start byte offset in the page (0 to PAGE_SIZE-1).

Return values

Float	or integer value read from the EEPROM.

Read a float or integer value from the EEPROM.

Parameters

page	Number of the start page. Range from 0 to PAGE_NUM-1.
offset	Start byte offset in the page. Range from 0 to PAGE_SIZE-1.

Return values

Float/integer	value.

9.3.2.4 EEPROM_Write()

```
void EEPROM_Write (
            uint16_t page,
            uint16_t offset,
            uint8_t * data,
            uint16_t size )
```

Write data to the EEPROM.

Parameters

page	Start page number (0 to PAGE_NUM-1).
offset	Start byte offset in the page (0 to PAGE_SIZE-1).
data	Pointer to the data to write in bytes.
size	Size of the data.

Return values

None	

9.3.2.5 EEPROM_Write_NUM()

```
void EEPROM_Write_NUM (
             uint16_t page,
             uint16_t offset,
             float data )
```

Write a float or integer value to the EEPROM.

Parameters

page	Start page number (0 to PAGE_NUM-1).
offset	Start byte offset in the page (0 to PAGE_SIZE-1).
fdata	Float or integer value to write.

Return values

None	

Write a float or integer value to the EEPROM.

Parameters

page	Number of the start page. Range from 0 to PAGE_NUM-1.
offset	Start byte offset in the page. Range from 0 to PAGE_SIZE-1.
data	Float/integer value that you want to write.

Return values

None.	

9.4 EEPROM.h

Go to the documentation of this file.

```
00001
00019 #ifndef INC_EEPROM_H_
00020 #define INC_EEPROM_H_
00021
00022 #include "stdint.h"
00023 #include "stm32f0xx_hal.h"
00024
00033 void EEPROM_Write(uint16_t page, uint16_t offset, uint8_t *data, uint16_t size);
00034
00043 void EEPROM_Read(uint16_t page, uint16_t offset, uint8_t *data, uint16_t size);
00044
00050 void EEPROM_PageErase(uint16_t page);
00051
00059 void EEPROM_Write_NUM(uint16_t page, uint16_t offset, float fdata);
00060
00067 float EEPROM_Read_NUM(uint16_t page, uint16_t offset);
00068
00069 #endif /* INC_EEPROM_H_ */
```

9.5 SENSEHAT INTEGRATION/Core/Inc/EEPROM.h File Reference

Using the HAL I2C Functions.

```
#include "stdint.h"
#include "stm32f0xx_hal.h"
```

Macros

- #define **PAGE_SIZE** 64
- #define **PAGE_NUM** 512

Functions

- void EEPROM_Write (uint16_t page, uint16_t offset, uint8_t *data, uint16_t size)

  Write data to the EEPROM.
- void EEPROM_Read (uint16_t page, uint16_t offset, uint8_t *data, uint16_t size)

  Read data from the EEPROM.
- void EEPROM_PageErase (uint16_t page)

  Erase a page in the EEPROM Memory.
- void EEPROM_Write_NUM (uint16_t page, uint16_t offset, float fdata)

  Write a float or integer value to the EEPROM.
- float EEPROM_Read_NUM (uint16_t page, uint16_t offset)

  Read a float or integer value from the EEPROM.

9.5.1 Detailed Description

Using the HAL I2C Functions.

Author

ControllersTech

Date

Feb 16, 2021

Attention

9.5.2 Function Documentation

9.5.2.1 EEPROM_PageErase()

```
void EEPROM_PageErase (
            uint16_t page )
```

Erase a page in the EEPROM Memory.

Parameters

page	Page number to erase.

Return values

None	

Parameters

page	Number of page to erase In order to erase multiple pages, just use this function in the for loop

Return values

None	

9.5.2.2 EEPROM_Read()

```
void EEPROM_Read (
            uint16_t page,
            uint16_t offset,
            uint8_t * data,
            uint16_t size )
```

Read data from the EEPROM.

Parameters

page	Start page number (0 to PAGE_NUM-1).
offset	Start byte offset in the page (0 to PAGE_SIZE-1).
data	Pointer to the data to read in bytes.
size	Size of the data.

Return values

None	

Parameters

page	Number of the start page. Range from 0 to PAGE_NUM-1.
offset	Start byte offset in the page. Range from 0 to PAGE_SIZE-1.
data	Pointer to the data to write in bytes.
size	Size of the data.

Return values

None.	

9.5.2.3 EEPROM_Read_NUM()

```
float EEPROM_Read_NUM (
            uint16_t page,
            uint16_t offset )
```

Read a float or integer value from the EEPROM.

Parameters

page	Start page number (0 to PAGE_NUM-1).
offset	Start byte offset in the page (0 to PAGE_SIZE-1).

Return values

Float	or integer value read from the EEPROM.

Read a float or integer value from the EEPROM.

Parameters

page	Number of the start page. Range from 0 to PAGE_NUM-1.
offset	Start byte offset in the page. Range from 0 to PAGE_SIZE-1.

Return values

Float/integer	value.

9.5.2.4 EEPROM_Write()

```
void EEPROM_Write (
            uint16_t page,
            uint16_t offset,
            uint8_t * data,
            uint16_t size )
```

Write data to the EEPROM.

Parameters

page	Start page number (0 to PAGE_NUM-1).
offset	Start byte offset in the page (0 to PAGE_SIZE-1).
data	Pointer to the data to write in bytes.
size	Size of the data.

Return values

None	

9.5.2.5 EEPROM_Write_NUM()

```
void EEPROM_Write_NUM (
```

```
            uint16_t page,
            uint16_t offset,
            float data )
```

Write a float or integer value to the EEPROM.

Parameters

page	Start page number (0 to PAGE_NUM-1).
offset	Start byte offset in the page (0 to PAGE_SIZE-1).
fdata	Float or integer value to write.

Return values

None	

Write a float or integer value to the EEPROM.

Parameters

page	Number of the start page. Range from 0 to PAGE_NUM-1.
offset	Start byte offset in the page. Range from 0 to PAGE_SIZE-1.
data	Float/integer value that you want to write.

Return values

None.	

9.6 EEPROM.h

Go to the documentation of this file.

```
00001
00019 #ifndef INC_EEPROM_H_
00020 #define INC_EEPROM_H_
00021
00022 #define PAGE_SIZE 64
00023 #define PAGE_NUM  512
00024
00025 #include "stdint.h"
00026 #include "stm32f0xx_hal.h"
00027
00036 void EEPROM_Write(uint16_t page, uint16_t offset, uint8_t *data, uint16_t size);
00037
00046 void EEPROM_Read(uint16_t page, uint16_t offset, uint8_t *data, uint16_t size);
00047
00053 void EEPROM_PageErase(uint16_t page);
00054
00062 void EEPROM_Write_NUM(uint16_t page, uint16_t offset, float fdata);
00063
00070 float EEPROM_Read_NUM(uint16_t page, uint16_t offset);
00071
00072 #endif /* INC_EEPROM_H_ */
```

9.7 LDR/ldr.c File Reference

LDR Library Source.

```
#include "ldr.h"
```

Functions

- void LDR_Init (void)

  Initialize the LDR with calibration constants.
- uint32_t LDR_ReadADC (ADC_HandleTypeDef *hadc)

  Read ADC value from the LDR.
- float LDR_ReadAnalogLightIntensity (ADC_HandleTypeDef *hadc)

  Read analog light intensity using the LDR.

9.7.1 Detailed Description

LDR Library Source.

Author

Travimadox Webb @position Embedded Software Engineer @company Imperium LLC

Date

6th of May 2023

9.7.2 Function Documentation

9.7.2.1 LDR_Init()

```
void LDR_Init (
            void  )
```

Initialize the LDR with calibration constants.

Initialize the LDR.

9.7.2.2 LDR_ReadADC()

```
uint32_t LDR_ReadADC (
            ADC_HandleTypeDef * hadc )
```

Read ADC value from the LDR.

Parameters

hadc	Pointer to an ADC_HandleTypeDef structure that contains the configuration information for the specified ADC

Returns

32-bit unsigned integer ADC value

9.7.2.3 LDR_ReadAnalogLightIntensity()

```
float LDR_ReadAnalogLightIntensity (
            ADC_HandleTypeDef * hadc )
```

Read analog light intensity using the LDR.

Parameters

hadc	Pointer to an ADC_HandleTypeDef structure that contains the configuration information for the specified ADC

Returns

Floating-point light intensity value

9.8 SENSEHAT INTEGRATION/Core/Src/ldr.c File Reference

LDR Library Source.

```
#include "ldr.h"
```

Functions

- void LDR_Init (void)

  Initialize the LDR with calibration constants.
- uint32_t LDR_ReadADC (ADC_HandleTypeDef *hadc)

  Read ADC value from the LDR.
- float LDR_ReadAnalogLightIntensity (ADC_HandleTypeDef *hadc)

  Read analog light intensity using the LDR.

9.8.1 Detailed Description

LDR Library Source.

Author

Travimadox Webb @position Embedded Software Engineer @company Imperium LLC

Date

6th of May 2023

9.8.2 Function Documentation

9.8.2.1 LDR_Init()

```
void LDR_Init (
            void  )
```

Initialize the LDR with calibration constants.

Initialize the LDR.

9.8.2.2 LDR_ReadADC()

```
uint32_t LDR_ReadADC (
            ADC_HandleTypeDef * hadc )
```

Read ADC value from the LDR.

Parameters

hadc	Pointer to an ADC_HandleTypeDef structure that contains the configuration information for the specified ADC

Returns

32-bit unsigned integer ADC value

9.8.2.3 LDR_ReadAnalogLightIntensity()

```
float LDR_ReadAnalogLightIntensity (
            ADC_HandleTypeDef * hadc )
```

Read analog light intensity using the LDR.

Parameters

hadc	Pointer to an ADC_HandleTypeDef structure that contains the configuration information for the specified ADC

Returns

Floating-point light intensity value

9.9 LDR/ldr.h File Reference

LDR Library Header.

```
#include "stm32f0xx_hal.h"
```

Functions

- void LDR_Init (void)
  Initialize the LDR.
- uint32_t LDR_ReadADC (ADC_HandleTypeDef ∗hadc)
  Read ADC value from the LDR.
- float LDR_ReadAnalogLightIntensity (ADC_HandleTypeDef ∗hadc)
  Read analog light intensity using the LDR.

9.9.1 Detailed Description

LDR Library Header.

Author

Travimadox Webb @position Embedded Software Engineer @company Imperium LLC

Date

6th of May 2023

9.9.2 Function Documentation

9.9.2.1 LDR_Init()

```
void LDR_Init (
            void  )
```

Initialize the LDR.

Initialize the LDR.

9.9.2.2 LDR_ReadADC()

```
uint32_t LDR_ReadADC (
            ADC_HandleTypeDef * hadc )
```

Read ADC value from the LDR.

Parameters

hadc	Pointer to an ADC_HandleTypeDef structure that contains the configuration information for the specified ADC

Returns

32-bit unsigned integer ADC value

9.9.2.3 LDR_ReadAnalogLightIntensity()

```
float LDR_ReadAnalogLightIntensity (
            ADC_HandleTypeDef * hadc )
```

Read analog light intensity using the LDR.

Parameters

hadc	Pointer to an ADC_HandleTypeDef structure that contains the configuration information for the specified ADC

Returns

Floating-point light intensity value

9.10 ldr.h

Go to the documentation of this file.

```
00001
00010 #ifndef LDR_H
00011 #define LDR_H
00012
00013 #include "stm32f0xx_hal.h"
00014
00018 void LDR_Init(void);
00019
00025 uint32_t LDR_ReadADC(ADC_HandleTypeDef *hadc);
00026
00032 float LDR_ReadAnalogLightIntensity(ADC_HandleTypeDef *hadc);
00033
00034 #endif // LDR_H
```

9.11 SENSEHAT INTEGRATION/Core/Inc/ldr.h File Reference

LDR Library Header.

```
#include "stm32f0xx_hal.h"
```

Functions

- void LDR_Init (void)

  Initialize the LDR.
- uint32_t LDR_ReadADC (ADC_HandleTypeDef *hadc)

  Read ADC value from the LDR.
- float LDR_ReadAnalogLightIntensity (ADC_HandleTypeDef *hadc)

  Read analog light intensity using the LDR.

9.11.1 Detailed Description

LDR Library Header.

Author

Travimadox Webb @position Embedded Software Engineer @company Imperium LLC

Date

6th of May 2023

9.11.2 Function Documentation

9.11.2.1 LDR_Init()

```
void LDR_Init (
            void  )
```

Initialize the LDR.

Initialize the LDR.

9.11.2.2 LDR_ReadADC()

```
uint32_t LDR_ReadADC (
            ADC_HandleTypeDef * hadc )
```

Read ADC value from the LDR.

Parameters

hadc	Pointer to an ADC_HandleTypeDef structure that contains the configuration information for the specified ADC

Returns

32-bit unsigned integer ADC value

9.11.2.3 LDR_ReadAnalogLightIntensity()

```
float LDR_ReadAnalogLightIntensity (
            ADC_HandleTypeDef * hadc )
```

Read analog light intensity using the LDR.

Parameters

hadc	Pointer to an ADC_HandleTypeDef structure that contains the configuration information for the specified ADC

Returns

Floating-point light intensity value

9.12 ldr.h

Go to the documentation of this file.

```
00001
00010 #ifndef LDR_H
00011 #define LDR_H
00012
00013 #include "stm32f0xx_hal.h"
00014
00018 void LDR_Init(void);
00019
00025 uint32_t LDR_ReadADC(ADC_HandleTypeDef *hadc);
00026
00032 float LDR_ReadAnalogLightIntensity(ADC_HandleTypeDef *hadc);
00033
00034 #endif // LDR_H
```

9.13 LTR303ALS/ltr303als.c File Reference

LTR-303ALS Ambient Light Sensor Library Implementation.

```
#include "ltr303als.h"
```

Functions

- HAL_StatusTypeDef LTR303ALS_Init (I2C_HandleTypeDef *hi2c, uint8_t integration_time, uint8_t measurement_rate)

  Initialize the LTR-303ALS Ambient Light Sensor.
- HAL_StatusTypeDef LTR303ALS_ReadLightIntensity (I2C_HandleTypeDef *hi2c, uint16_t *ch0, uint16_t *ch1)

  Read light intensity from the LTR-303ALS Ambient Light Sensor.

9.13.1 Detailed Description

LTR-303ALS Ambient Light Sensor Library Implementation.

Author

Travimadox Webb @position Embedded Software Engineer @company Imperium LLC

Date

6th of May 2023

9.13.2 Function Documentation

9.13.2.1 LTR303ALS_Init()

```
HAL_StatusTypeDef LTR303ALS_Init (
            I2C_HandleTypeDef * hi2c,
            uint8_t integration_time,
            uint8_t measurement_rate )
```

Initialize the LTR-303ALS Ambient Light Sensor.

Initialize the LTR-303ALS sensor.

Parameters

hi2c	Pointer to an I2C_HandleTypeDef structure that contains the configuration information for the specified I2C
integration_time	Integration time for the LTR-303ALS
measurement_rate	Measurement rate for the LTR-303ALS

Returns

HAL status (HAL_OK if successful, HAL_ERROR otherwise)

9.13.2.2 LTR303ALS_ReadLightIntensity()

```
HAL_StatusTypeDef LTR303ALS_ReadLightIntensity (
            I2C_HandleTypeDef * hi2c,
            uint16_t * ch0,
            uint16_t * ch1 )
```

Read light intensity from the LTR-303ALS Ambient Light Sensor.

Read light intensity data from the LTR-303ALS sensor.

Parameters

hi2c	Pointer to an I2C_HandleTypeDef structure that contains the configuration information for the specified I2C
ch0	Pointer to a uint16_t variable to store the Channel 0 data
ch1	Pointer to a uint16_t variable to store the Channel 1 data

Returns

HAL status (HAL_OK if successful, HAL_ERROR otherwise)

9.14 SENSEHAT INTEGRATION/Core/Src/ltr303als.c File Reference

LTR-303ALS Ambient Light Sensor Library Implementation.

```
#include "ltr303als.h"
```

Functions

- HAL_StatusTypeDef LTR303ALS_Init (I2C_HandleTypeDef *hi2c, uint8_t integration_time, uint8_t measurement_rate)

  Initialize the LTR-303ALS Ambient Light Sensor.
- HAL_StatusTypeDef LTR303ALS_ReadLightIntensity (I2C_HandleTypeDef *hi2c, uint16_t *ch0, uint16_t *ch1)

  Read light intensity from the LTR-303ALS Ambient Light Sensor.

9.14.1 Detailed Description

LTR-303ALS Ambient Light Sensor Library Implementation.

Author

Travimadox Webb @position Embedded Software Engineer @company Imperium LLC

Date

6th of May 2023

9.14.2 Function Documentation

9.14.2.1 LTR303ALS_Init()

```
HAL_StatusTypeDef LTR303ALS_Init (
            I2C_HandleTypeDef * hi2c,
            uint8_t integration_time,
            uint8_t measurement_rate )
```

Initialize the LTR-303ALS Ambient Light Sensor.

Initialize the LTR-303ALS sensor.

Parameters

hi2c	Pointer to an I2C_HandleTypeDef structure that contains the configuration information for the specified I2C
integration_time	Integration time for the LTR-303ALS
measurement_rate	Measurement rate for the LTR-303ALS

Returns

HAL status (HAL_OK if successful, HAL_ERROR otherwise)

9.14.2.2 LTR303ALS_ReadLightIntensity()

```
HAL_StatusTypeDef LTR303ALS_ReadLightIntensity (
            I2C_HandleTypeDef * hi2c,
            uint16_t * ch0,
            uint16_t * ch1 )
```

Read light intensity from the LTR-303ALS Ambient Light Sensor.

Read light intensity data from the LTR-303ALS sensor.

Parameters

hi2c	Pointer to an I2C_HandleTypeDef structure that contains the configuration information for the specified I2C
ch0	Pointer to a uint16_t variable to store the Channel 0 data
ch1	Pointer to a uint16_t variable to store the Channel 1 data

Returns

HAL status (HAL_OK if successful, HAL_ERROR otherwise)

9.15 LTR303ALS/ltr303als.h File Reference

LTR-303ALS Ambient Light Sensor Library Header.

```
#include "stm32f0xx_hal.h"
```

Macros

- #define **LTR303ALS_I2C_ADDRESS** 0x29

Functions

- HAL_StatusTypeDef LTR303ALS_Init (I2C_HandleTypeDef *hi2c, uint8_t integration_time, uint8_t measurement_rate)

  Initialize the LTR-303ALS sensor.
- HAL_StatusTypeDef LTR303ALS_ReadLightIntensity (I2C_HandleTypeDef *hi2c, uint16_t *ch0, uint16_t *ch1)

  Read light intensity data from the LTR-303ALS sensor.

9.15.1 Detailed Description

LTR-303ALS Ambient Light Sensor Library Header.

Author

Travimadox Webb @position Embedded Software Engineer @company Imperium LLC

Date

7th of May 2023

9.15.2 Function Documentation

9.15.2.1 LTR303ALS_Init()

```
HAL_StatusTypeDef LTR303ALS_Init (
            I2C_HandleTypeDef * hi2c,
            uint8_t integration_time,
            uint8_t measurement_rate )
```

Initialize the LTR-303ALS sensor.

Parameters

hi2c	Pointer to the I2C handle.
integration_time	Integration time (in ms) for the sensor.
measurement_rate	Measurement rate (in ms) for the sensor.

Returns

HAL status.

Initialize the LTR-303ALS sensor.

Parameters

hi2c	Pointer to an I2C_HandleTypeDef structure that contains the configuration information for the specified I2C
integration_time	Integration time for the LTR-303ALS
measurement_rate	Measurement rate for the LTR-303ALS

Returns

HAL status (HAL_OK if successful, HAL_ERROR otherwise)

9.15.2.2 LTR303ALS_ReadLightIntensity()

```
HAL_StatusTypeDef LTR303ALS_ReadLightIntensity (
            I2C_HandleTypeDef * hi2c,
            uint16_t * ch0,
            uint16_t * ch1 )
```

Read light intensity data from the LTR-303ALS sensor.

Parameters

hi2c	Pointer to the I2C handle.
ch0	Pointer to store the channel 0 data.
ch1	Pointer to store the channel 1 data.

Returns

HAL status.

Read light intensity data from the LTR-303ALS sensor.

Parameters

hi2c	Pointer to an I2C_HandleTypeDef structure that contains the configuration information for the specified I2C
ch0	Pointer to a uint16_t variable to store the Channel 0 data
ch1	Pointer to a uint16_t variable to store the Channel 1 data

Returns

HAL status (HAL_OK if successful, HAL_ERROR otherwise)

9.16 ltr303als.h

Go to the documentation of this file.

```
00001
00010 #ifndef LTR303ALS_H
00011 #define LTR303ALS_H
00012
00013 #include "stm32f0xx_hal.h"
00014
00015 // LTR-303ALS I2C address (default: 0x29)
00016 #define LTR303ALS_I2C_ADDRESS 0x29
00017
00025 HAL_StatusTypeDef LTR303ALS_Init(I2C_HandleTypeDef *hi2c, uint8_t integration_time, uint8_t
      measurement_rate);
00026
00034 HAL_StatusTypeDef LTR303ALS_ReadLightIntensity(I2C_HandleTypeDef *hi2c, uint16_t *ch0, uint16_t *ch1);
00035
00036 #endif // LTR303ALS_H
```

9.17 SENSEHAT INTEGRATION/Core/Inc/ltr303als.h File Reference

LTR-303ALS Ambient Light Sensor Library Header.

`#include "stm32f0xx_hal.h"`

Macros

- #define **LTR303ALS_I2C_ADDRESS** 0x29

Functions

- HAL_StatusTypeDef LTR303ALS_Init (I2C_HandleTypeDef *hi2c, uint8_t integration_time, uint8_t measurement_rate)

  Initialize the LTR-303ALS sensor.
- HAL_StatusTypeDef LTR303ALS_ReadLightIntensity (I2C_HandleTypeDef *hi2c, uint16_t *ch0, uint16_t *ch1)

  Read light intensity data from the LTR-303ALS sensor.

9.17.1 Detailed Description

LTR-303ALS Ambient Light Sensor Library Header.

Author

Travimadox Webb @position Embedded Software Engineer @company Imperium LLC

Date

7th of May 2023

9.17.2 Function Documentation

9.17.2.1 LTR303ALS_Init()

```
HAL_StatusTypeDef LTR303ALS_Init (
            I2C_HandleTypeDef * hi2c,
            uint8_t integration_time,
            uint8_t measurement_rate )
```

Initialize the LTR-303ALS sensor.

Parameters

hi2c	Pointer to the I2C handle.
integration_time	Integration time (in ms) for the sensor.
measurement_rate	Measurement rate (in ms) for the sensor.

Returns

HAL status.

Initialize the LTR-303ALS sensor.

Parameters

hi2c	Pointer to an I2C_HandleTypeDef structure that contains the configuration information for the specified I2C
integration_time	Integration time for the LTR-303ALS
measurement_rate	Measurement rate for the LTR-303ALS

Returns

HAL status (HAL_OK if successful, HAL_ERROR otherwise)

9.17.2.2 LTR303ALS_ReadLightIntensity()

```
HAL_StatusTypeDef LTR303ALS_ReadLightIntensity (
            I2C_HandleTypeDef * hi2c,
            uint16_t * ch0,
            uint16_t * ch1 )
```

Read light intensity data from the LTR-303ALS sensor.

Parameters

hi2c	Pointer to the I2C handle.
ch0	Pointer to store the channel 0 data.
ch1	Pointer to store the channel 1 data.

Returns

HAL status.

Read light intensity data from the LTR-303ALS sensor.

Parameters

hi2c	Pointer to an I2C_HandleTypeDef structure that contains the configuration information for the specified I2C
ch0	Pointer to a uint16_t variable to store the Channel 0 data
ch1	Pointer to a uint16_t variable to store the Channel 1 data

Returns

HAL status (HAL_OK if successful, HAL_ERROR otherwise)

9.18 ltr303als.h

Go to the documentation of this file.

```
00001
00010 #ifndef LTR303ALS_H
00011 #define LTR303ALS_H
00012
00013 #include "stm32f0xx_hal.h"
00014
00015 // LTR-303ALS I2C address (default: 0x29)
00016 #define LTR303ALS_I2C_ADDRESS 0x29
00017
00025 HAL_StatusTypeDef LTR303ALS_Init(I2C_HandleTypeDef *hi2c, uint8_t integration_time, uint8_t
      measurement_rate);
00026
00034 HAL_StatusTypeDef LTR303ALS_ReadLightIntensity(I2C_HandleTypeDef *hi2c, uint16_t *ch0, uint16_t *ch1);
00035
00036 #endif // LTR303ALS_H
```

9.19 RTC/rtc.c File Reference

RTC Library Source.

```
#include "rtc.h"
```

Functions

- HAL_StatusTypeDef RTC_SetTime (RTC_HandleTypeDef ∗hrtc, uint8_t hours, uint8_t minutes, uint8_t seconds)

  Set the RTC time.
- HAL_StatusTypeDef RTC_GetTime (RTC_HandleTypeDef ∗hrtc, uint8_t ∗hours, uint8_t ∗minutes, uint8_t ∗seconds)

  Get the RTC time.
- void RTC_SetDate (RTC_HandleTypeDef ∗hrtc, uint8_t day, uint8_t month, uint8_t year)

  Set the RTC date.
- void RTC_GetDate (RTC_HandleTypeDef ∗hrtc, uint8_t ∗day, uint8_t ∗month, uint8_t ∗year)

  Get the RTC date.

9.19.1 Detailed Description

RTC Library Source.

Author

Travimadox Webb @position Embedded Software Engineer @company Imperium LLC

Date

7th May 2023

Note

IMPORTANT: Initialize the RTC module using STM32CubeIDE GUI. Configure RTC clock source and enable RTC in the main initialization code or in the Clock Configuration function (SystemClock_Config).

9.19.2 Function Documentation

9.19.2.1 RTC_GetDate()

```
void RTC_GetDate (
            RTC_HandleTypeDef * hrtc,
            uint8_t * day,
            uint8_t * month,
            uint8_t * year )
```

Get the RTC date.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
day	Pointer to store the day of the month
month	Pointer to store the month
year	Pointer to store the year

9.19.2.2 RTC_GetTime()

```
HAL_StatusTypeDef RTC_GetTime (
            RTC_HandleTypeDef * hrtc,
            uint8_t * hours,
            uint8_t * minutes,
            uint8_t * seconds )
```

Get the RTC time.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
hours	Pointer to an uint8_t variable to store the hours value
minutes	Pointer to an uint8_t variable to store the minutes value
seconds	Pointer to an uint8_t variable to store the seconds value

Returns

HAL status (HAL_OK, HAL_ERROR, HAL_BUSY, or HAL_TIMEOUT)

9.19.2.3 RTC_SetDate()

```
void RTC_SetDate (
            RTC_HandleTypeDef * hrtc,
            uint8_t day,
            uint8_t month,
            uint8_t year )
```

Set the RTC date.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
day	The day of the month
month	The month
year	The year (from 0 to 99)

9.19.2.4 RTC_SetTime()

```
HAL_StatusTypeDef RTC_SetTime (
            RTC_HandleTypeDef * hrtc,
            uint8_t hours,
            uint8_t minutes,
            uint8_t seconds )
```

Set the RTC time.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
hours	Hours value to set (0-23)
minutes	Minutes value to set (0-59)
seconds	Seconds value to set (0-59)

Returns

HAL status (HAL_OK, HAL_ERROR, HAL_BUSY, or HAL_TIMEOUT)

9.20 SENSEHAT INTEGRATION/Core/Src/rtc.c File Reference

RTC Library Source.

```
#include "rtc.h"
```

Functions

- HAL_StatusTypeDef RTC_SetTime (RTC_HandleTypeDef ∗hrtc, uint8_t hours, uint8_t minutes, uint8_t seconds)

  Set the RTC time.
- HAL_StatusTypeDef RTC_GetTime (RTC_HandleTypeDef ∗hrtc, uint8_t ∗hours, uint8_t ∗minutes, uint8_t ∗seconds)

  Get the RTC time.
- void RTC_SetDate (RTC_HandleTypeDef ∗hrtc, uint8_t day, uint8_t month, uint8_t year)

  Set the RTC date.
- void RTC_GetDate (RTC_HandleTypeDef ∗hrtc, uint8_t ∗day, uint8_t ∗month, uint8_t ∗year)

  Get the RTC date.

9.20.1 Detailed Description

RTC Library Source.

Author

Travimadox Webb @position Embedded Software Engineer @company Imperium LLC

Date

7th May 2023

Note

IMPORTANT: Initialize the RTC module using STM32CubeIDE GUI. Configure RTC clock source and enable RTC in the main initialization code or in the Clock Configuration function (SystemClock_Config).

9.20.2 Function Documentation

9.20.2.1 RTC_GetDate()

```
void RTC_GetDate (
            RTC_HandleTypeDef * hrtc,
            uint8_t * day,
            uint8_t * month,
            uint8_t * year )
```

Get the RTC date.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
day	Pointer to store the day of the month
month	Pointer to store the month
year	Pointer to store the year

9.20.2.2 RTC_GetTime()

```
HAL_StatusTypeDef RTC_GetTime (
            RTC_HandleTypeDef * hrtc,
            uint8_t * hours,
            uint8_t * minutes,
            uint8_t * seconds )
```

Get the RTC time.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
hours	Pointer to an uint8_t variable to store the hours value
minutes	Pointer to an uint8_t variable to store the minutes value
seconds	Pointer to an uint8_t variable to store the seconds value

Returns

HAL status (HAL_OK, HAL_ERROR, HAL_BUSY, or HAL_TIMEOUT)

9.20.2.3 RTC_SetDate()

```
void RTC_SetDate (
            RTC_HandleTypeDef * hrtc,
            uint8_t day,
            uint8_t month,
            uint8_t year )
```

Set the RTC date.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
day	The day of the month
month	The month
year	The year (from 0 to 99)

9.20.2.4 RTC_SetTime()

```
HAL_StatusTypeDef RTC_SetTime (
            RTC_HandleTypeDef * hrtc,
```

```
            uint8_t hours,
            uint8_t minutes,
            uint8_t seconds )
```

Set the RTC time.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
hours	Hours value to set (0-23)
minutes	Minutes value to set (0-59)
seconds	Seconds value to set (0-59)

Returns

HAL status (HAL_OK, HAL_ERROR, HAL_BUSY, or HAL_TIMEOUT)

9.21 RTC/rtc.h File Reference

RTC Library Header.

```
#include "stm32f0xx_hal.h"
```

Functions

- HAL_StatusTypeDef RTC_SetTime (RTC_HandleTypeDef ∗hrtc, uint8_t hours, uint8_t minutes, uint8_t seconds)

  Set the RTC time.
- HAL_StatusTypeDef RTC_GetTime (RTC_HandleTypeDef ∗hrtc, uint8_t ∗hours, uint8_t ∗minutes, uint8_t ∗seconds)

  Get the RTC time.
- void RTC_SetDate (RTC_HandleTypeDef ∗hrtc, uint8_t day, uint8_t month, uint8_t year)

  Set the RTC date.
- void RTC_GetDate (RTC_HandleTypeDef ∗hrtc, uint8_t ∗day, uint8_t ∗month, uint8_t ∗year)

  Get the RTC date.

9.21.1 Detailed Description

RTC Library Header.

Author

Travimadox Webb @position Embedded Software Engineer @company Imperium LLC

Date

7th May 2023

9.21.2 Function Documentation

9.21.2.1 RTC_GetDate()

```
void RTC_GetDate (
            RTC_HandleTypeDef * hrtc,
            uint8_t * day,
            uint8_t * month,
            uint8_t * year )
```

Get the RTC date.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
day	Pointer to store the day of the month
month	Pointer to store the month
year	Pointer to store the year

9.21.2.2 RTC_GetTime()

```
HAL_StatusTypeDef RTC_GetTime (
            RTC_HandleTypeDef * hrtc,
            uint8_t * hours,
            uint8_t * minutes,
            uint8_t * seconds )
```

Get the RTC time.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
hours	Pointer to an uint8_t variable to store the hours value
minutes	Pointer to an uint8_t variable to store the minutes value
seconds	Pointer to an uint8_t variable to store the seconds value

Returns

HAL status (HAL_OK, HAL_ERROR, HAL_BUSY, or HAL_TIMEOUT)

9.21.2.3 RTC_SetDate()

```
void RTC_SetDate (
             RTC_HandleTypeDef * hrtc,
             uint8_t day,
             uint8_t month,
             uint8_t year )
```

Set the RTC date.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
day	The day of the month
month	The month
year	The year (from 0 to 99)

9.21.2.4 RTC_SetTime()

```
HAL_StatusTypeDef RTC_SetTime (
             RTC_HandleTypeDef * hrtc,
             uint8_t hours,
             uint8_t minutes,
             uint8_t seconds )
```

Set the RTC time.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
hours	Hours value to set (0-23)
minutes	Minutes value to set (0-59)
seconds	Seconds value to set (0-59)

Returns

HAL status (HAL_OK, HAL_ERROR, HAL_BUSY, or HAL_TIMEOUT)

9.22 rtc.h

Go to the documentation of this file.

```
00001
00010 #ifndef __RTC_H
00011 #define __RTC_H
00012
00013 #include "stm32f0xx_hal.h"
00014
```

```
00023 HAL_StatusTypeDef RTC_SetTime(RTC_HandleTypeDef *hrtc, uint8_t hours, uint8_t minutes, uint8_t
      seconds);
00024
00033 HAL_StatusTypeDef RTC_GetTime(RTC_HandleTypeDef *hrtc, uint8_t *hours, uint8_t *minutes, uint8_t
      *seconds);
00034
00035
00044 void RTC_SetDate(RTC_HandleTypeDef *hrtc, uint8_t day, uint8_t month, uint8_t year);
00053 void RTC_GetDate(RTC_HandleTypeDef *hrtc, uint8_t *day, uint8_t *month, uint8_t *year);
00054
00055 #endif // __RTC_H
```

9.23 SENSEHAT INTEGRATION/Core/Inc/rtc.h File Reference

RTC Library Header.

```
#include "stm32f0xx_hal.h"
```

Functions

- HAL_StatusTypeDef RTC_SetTime (RTC_HandleTypeDef ∗hrtc, uint8_t hours, uint8_t minutes, uint8_t seconds)

  Set the RTC time.
- HAL_StatusTypeDef RTC_GetTime (RTC_HandleTypeDef ∗hrtc, uint8_t ∗hours, uint8_t ∗minutes, uint8_t ∗seconds)

  Get the RTC time.
- void RTC_SetDate (RTC_HandleTypeDef ∗hrtc, uint8_t day, uint8_t month, uint8_t year)

  Set the RTC date.
- void RTC_GetDate (RTC_HandleTypeDef ∗hrtc, uint8_t ∗day, uint8_t ∗month, uint8_t ∗year)

  Get the RTC date.

9.23.1 Detailed Description

RTC Library Header.

Author

Travimadox Webb @position Embedded Software Engineer @company Imperium LLC

Date

7th May 2023

9.23.2 Function Documentation

9.23.2.1 RTC_GetDate()

```
void RTC_GetDate (
            RTC_HandleTypeDef * hrtc,
            uint8_t * day,
            uint8_t * month,
            uint8_t * year )
```

Get the RTC date.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
day	Pointer to store the day of the month
month	Pointer to store the month
year	Pointer to store the year

9.23.2.2 RTC_GetTime()

```
HAL_StatusTypeDef RTC_GetTime (
            RTC_HandleTypeDef * hrtc,
            uint8_t * hours,
            uint8_t * minutes,
            uint8_t * seconds )
```

Get the RTC time.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
hours	Pointer to an uint8_t variable to store the hours value
minutes	Pointer to an uint8_t variable to store the minutes value
seconds	Pointer to an uint8_t variable to store the seconds value

Returns

HAL status (HAL_OK, HAL_ERROR, HAL_BUSY, or HAL_TIMEOUT)

9.23.2.3 RTC_SetDate()

```
void RTC_SetDate (
            RTC_HandleTypeDef * hrtc,
            uint8_t day,
            uint8_t month,
            uint8_t year )
```

Set the RTC date.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
day	The day of the month
month	The month
year	The year (from 0 to 99)

9.23.2.4 RTC_SetTime()

```
HAL_StatusTypeDef RTC_SetTime (
            RTC_HandleTypeDef * hrtc,
            uint8_t hours,
            uint8_t minutes,
            uint8_t seconds )
```

Set the RTC time.

Parameters

hrtc	Pointer to an RTC_HandleTypeDef structure that contains the configuration information for the specified RTC
hours	Hours value to set (0-23)
minutes	Minutes value to set (0-59)
seconds	Seconds value to set (0-59)

Returns

HAL status (HAL_OK, HAL_ERROR, HAL_BUSY, or HAL_TIMEOUT)

9.24 rtc.h

Go to the documentation of this file.

```
00001
00010 #ifndef __RTC_H
00011 #define __RTC_H
00012
00013 #include "stm32f0xx_hal.h"
00014
00023 HAL_StatusTypeDef RTC_SetTime(RTC_HandleTypeDef *hrtc, uint8_t hours, uint8_t minutes, uint8_t
      seconds);
00024
00033 HAL_StatusTypeDef RTC_GetTime(RTC_HandleTypeDef *hrtc, uint8_t *hours, uint8_t *minutes, uint8_t
      *seconds);
00034
00035
00044 void RTC_SetDate(RTC_HandleTypeDef *hrtc, uint8_t day, uint8_t month, uint8_t year);
00053 void RTC_GetDate(RTC_HandleTypeDef *hrtc, uint8_t *day, uint8_t *month, uint8_t *year);
00054
00055 #endif // __RTC_H
```

9.25 SENSEHAT INTEGRATION/Core/Inc/main.h File Reference

: Header for main.c file. This file contains the common defines of the application.

```
#include "stm32f0xx_hal.h"
```

Functions

- void Error_Handler (void)

  This function is executed in case of error occurrence.

9.25.1 Detailed Description

: Header for main.c file. This file contains the common defines of the application.

Attention

9.25.2 Function Documentation

9.25.2.1 Error_Handler()

```
void Error_Handler (
             void  )
```

This function is executed in case of error occurrence.

Return values

None	

9.26 main.h

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Define to prevent recursive inclusion -------------------------------------*/
00022 #ifndef __MAIN_H
00023 #define __MAIN_H
00024
00025 #ifdef __cplusplus
00026 extern "C" {
00027 #endif
00028
00029 /* Includes ------------------------------------------------------------------*/
00030 #include "stm32f0xx_hal.h"
00031
00032 /* Private includes ----------------------------------------------------------*/
00033 /* USER CODE BEGIN Includes */
00034
00035 /* USER CODE END Includes */
00036
00037 /* Exported types ------------------------------------------------------------*/
00038 /* USER CODE BEGIN ET */
00039
00040 /* USER CODE END ET */
00041
00042 /* Exported constants --------------------------------------------------------*/
00043 /* USER CODE BEGIN EC */
```

```
00044
00045 /* USER CODE END EC */
00046
00047 /* Exported macro ------------------------------------------------------------*/
00048 /* USER CODE BEGIN EM */
00049
00050 /* USER CODE END EM */
00051
00052 /* Exported functions prototypes ---------------------------------------------*/
00053 void Error_Handler(void);
00054
00055 /* USER CODE BEGIN EFP */
00056
00057 /* USER CODE END EFP */
00058
00059 /* Private defines -----------------------------------------------------------*/
00060
00061 /* USER CODE BEGIN Private defines */
00062
00063 /* USER CODE END Private defines */
00064
00065 #ifdef __cplusplus
00066 }
00067 #endif
00068
00069 #endif /* __MAIN_H */
```

9.27 SENSEHAT INTEGRATION/Core/Inc/stm32f0xx_hal_conf.h File Reference

HAL configuration file.

```
#include "stm32f0xx_hal_rcc.h"
#include "stm32f0xx_hal_gpio.h"
#include "stm32f0xx_hal_exti.h"
#include "stm32f0xx_hal_dma.h"
#include "stm32f0xx_hal_cortex.h"
#include "stm32f0xx_hal_adc.h"
#include "stm32f0xx_hal_flash.h"
#include "stm32f0xx_hal_i2c.h"
#include "stm32f0xx_hal_pwr.h"
#include "stm32f0xx_hal_rtc.h"
#include "stm32f0xx_hal_tim.h"
#include "stm32f0xx_hal_uart.h"
```

Macros

- #define **HAL_MODULE_ENABLED**

  This is the list of modules to be used in the HAL driver.
- #define **HAL_ADC_MODULE_ENABLED**
- #define **HAL_RTC_MODULE_ENABLED**
- #define **HAL_TIM_MODULE_ENABLED**
- #define **HAL_UART_MODULE_ENABLED**
- #define **HAL_CORTEX_MODULE_ENABLED**
- #define **HAL_DMA_MODULE_ENABLED**
- #define **HAL_FLASH_MODULE_ENABLED**
- #define **HAL_GPIO_MODULE_ENABLED**
- #define **HAL_EXTI_MODULE_ENABLED**
- #define **HAL_PWR_MODULE_ENABLED**
- #define **HAL_RCC_MODULE_ENABLED**

- #define **HAL_I2C_MODULE_ENABLED**
- #define HSE_VALUE ((uint32_t)8000000)

  Adjust the value of External High Speed oscillator (HSE) used in your application. This value is used by the RCC HAL module to compute the system frequency (when HSE is used as system clock source, directly or through the PLL).
- #define HSE_STARTUP_TIMEOUT ((uint32_t)100)

  In the following line adjust the External High Speed oscillator (HSE) Startup Timeout value.
- #define HSI_VALUE ((uint32_t)8000000)

  Internal High Speed oscillator (HSI) value. This value is used by the RCC HAL module to compute the system frequency (when HSI is used as system clock source, directly or through the PLL).
- #define HSI_STARTUP_TIMEOUT ((uint32_t)5000)

  In the following line adjust the Internal High Speed oscillator (HSI) Startup Timeout value.
- #define HSI14_VALUE ((uint32_t)14000000)

  Internal High Speed oscillator for ADC (HSI14) value.
- #define HSI48_VALUE ((uint32_t)48000000)

  Internal High Speed oscillator for USB (HSI48) value.
- #define **LSI_VALUE** ((uint32_t)40000)

  Internal Low Speed oscillator (LSI) value.
- #define LSE_VALUE ((uint32_t)32768)

  External Low Speed oscillator (LSI) value.
- #define LSE_STARTUP_TIMEOUT ((uint32_t)5000)

  Time out for LSE start up value in ms.
- #define VDD_VALUE ((uint32_t)3300)

  This is the HAL system configuration section.
- #define TICK_INT_PRIORITY ((uint32_t)3)
- #define **USE_RTOS** 0
- #define **PREFETCH_ENABLE** 1
- #define **INSTRUCTION_CACHE_ENABLE** 0
- #define **DATA_CACHE_ENABLE** 0
- #define **USE_SPI_CRC** 0U
- #define **USE_HAL_ADC_REGISTER_CALLBACKS** 0U /∗ ADC register callback disabled ∗/
- #define **USE_HAL_CAN_REGISTER_CALLBACKS** 0U /∗ CAN register callback disabled ∗/
- #define **USE_HAL_COMP_REGISTER_CALLBACKS** 0U /∗ COMP register callback disabled ∗/
- #define **USE_HAL_CEC_REGISTER_CALLBACKS** 0U /∗ CEC register callback disabled ∗/
- #define **USE_HAL_DAC_REGISTER_CALLBACKS** 0U /∗ DAC register callback disabled ∗/
- #define **USE_HAL_I2C_REGISTER_CALLBACKS** 0U /∗ I2C register callback disabled ∗/
- #define **USE_HAL_SMBUS_REGISTER_CALLBACKS** 0U /∗ SMBUS register callback disabled ∗/
- #define **USE_HAL_UART_REGISTER_CALLBACKS** 0U /∗ UART register callback disabled ∗/
- #define **USE_HAL_USART_REGISTER_CALLBACKS** 0U /∗ USART register callback disabled ∗/
- #define **USE_HAL_IRDA_REGISTER_CALLBACKS** 0U /∗ IRDA register callback disabled ∗/
- #define **USE_HAL_SMARTCARD_REGISTER_CALLBACKS** 0U /∗ SMARTCARD register callback disabled ∗/
- #define **USE_HAL_WWDG_REGISTER_CALLBACKS** 0U /∗ WWDG register callback disabled ∗/
- #define **USE_HAL_RTC_REGISTER_CALLBACKS** 0U /∗ RTC register callback disabled ∗/
- #define **USE_HAL_SPI_REGISTER_CALLBACKS** 0U /∗ SPI register callback disabled ∗/
- #define **USE_HAL_I2S_REGISTER_CALLBACKS** 0U /∗ I2S register callback disabled ∗/
- #define **USE_HAL_TIM_REGISTER_CALLBACKS** 0U /∗ TIM register callback disabled ∗/
- #define **USE_HAL_TSC_REGISTER_CALLBACKS** 0U /∗ TSC register callback disabled ∗/
- #define **USE_HAL_PCD_REGISTER_CALLBACKS** 0U /∗ PCD register callback disabled ∗/
- #define assert_param(expr) ((void)0U)

  Uncomment the line below to expanse the "assert_param" macro in the HAL drivers code.

9.27.1 Detailed Description

HAL configuration file.

Attention

9.27.2 Macro Definition Documentation

9.27.2.1 assert_param

```
#define assert_param(
             expr ) ((void)0U)
```

Uncomment the line below to expanse the "assert_param" macro in the HAL drivers code.

Include module's header file

9.27.2.2 HSE_STARTUP_TIMEOUT

```
#define HSE_STARTUP_TIMEOUT ((uint32_t)100)
```

In the following line adjust the External High Speed oscillator (HSE) Startup Timeout value.

Time out for HSE start up, in ms

9.27.2.3 HSE_VALUE

```
#define HSE_VALUE ((uint32_t)8000000)
```

Adjust the value of External High Speed oscillator (HSE) used in your application. This value is used by the RCC HAL module to compute the system frequency (when HSE is used as system clock source, directly or through the PLL).

Value of the External oscillator in Hz

9.27.2.4 HSI14_VALUE

```
#define HSI14_VALUE ((uint32_t)14000000)
```

Internal High Speed oscillator for ADC (HSI14) value.

Value of the Internal High Speed oscillator for ADC in Hz. The real value may vary depending on the variations in voltage and temperature.

9.27.2.5 HSI48_VALUE

```
#define HSI48_VALUE ((uint32_t)48000000)
```

Internal High Speed oscillator for USB (HSI48) value.

Value of the Internal High Speed oscillator for USB in Hz. The real value may vary depending on the variations in voltage and temperature.

9.27.2.6 HSI_STARTUP_TIMEOUT

```
#define HSI_STARTUP_TIMEOUT ((uint32_t)5000)
```

In the following line adjust the Internal High Speed oscillator (HSI) Startup Timeout value.

Time out for HSI start up

9.27.2.7 HSI_VALUE

```
#define HSI_VALUE ((uint32_t)8000000)
```

Internal High Speed oscillator (HSI) value. This value is used by the RCC HAL module to compute the system frequency (when HSI is used as system clock source, directly or through the PLL).

Value of the Internal oscillator in Hz

9.27.2.8 LSE_STARTUP_TIMEOUT

```
#define LSE_STARTUP_TIMEOUT ((uint32_t)5000)
```

Time out for LSE start up value in ms.

Time out for LSE start up, in ms

9.27.2.9 LSE_VALUE

```
#define LSE_VALUE ((uint32_t)32768)
```

External Low Speed oscillator (LSI) value.

< Value of the Internal Low Speed oscillator in Hz The real value may vary depending on the variations in voltage and temperature.
Value of the External Low Speed oscillator in Hz

9.27.2.10 TICK_INT_PRIORITY

```
#define TICK_INT_PRIORITY ((uint32_t)3)
```

tick interrupt priority (lowest by default)

9.27.2.11 VDD_VALUE

```
#define VDD_VALUE ((uint32_t)3300)
```

This is the HAL system configuration section.

Value of VDD in mv

9.28 stm32f0xx_hal_conf.h

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00018 /* USER CODE END Header */
00019
00020 /* Define to prevent recursive inclusion -------------------------------------*/
00021 #ifndef __STM32F0xx_HAL_CONF_H
00022 #define __STM32F0xx_HAL_CONF_H
00023
00024 #ifdef __cplusplus
00025  extern "C" {
00026 #endif
00027
00028 /* Exported types ------------------------------------------------------------*/
00029 /* Exported constants --------------------------------------------------------*/
00030
00031 /* ########################## Module Selection ############################## */
00035 #define HAL_MODULE_ENABLED
00036   #define HAL_ADC_MODULE_ENABLED
00037 /*#define HAL_CRYP_MODULE_ENABLED   */
00038 /*#define HAL_CAN_MODULE_ENABLED   */
00039 /*#define HAL_CEC_MODULE_ENABLED   */
00040 /*#define HAL_COMP_MODULE_ENABLED   */
00041 /*#define HAL_CRC_MODULE_ENABLED   */
00042 /*#define HAL_CRYP_MODULE_ENABLED   */
00043 /*#define HAL_TSC_MODULE_ENABLED   */
00044 /*#define HAL_DAC_MODULE_ENABLED   */
00045 /*#define HAL_I2S_MODULE_ENABLED   */
00046 /*#define HAL_IWDG_MODULE_ENABLED   */
00047 /*#define HAL_LCD_MODULE_ENABLED   */
00048 /*#define HAL_LPTIM_MODULE_ENABLED   */
00049 /*#define HAL_RNG_MODULE_ENABLED   */
00050 #define HAL_RTC_MODULE_ENABLED
00051 /*#define HAL_SPI_MODULE_ENABLED   */
00052 #define HAL_TIM_MODULE_ENABLED
00053 #define HAL_UART_MODULE_ENABLED
00054 /*#define HAL_USART_MODULE_ENABLED   */
00055 /*#define HAL_IRDA_MODULE_ENABLED   */
```

```
00056 /*#define HAL_SMARTCARD_MODULE_ENABLED   */
00057 /*#define HAL_SMBUS_MODULE_ENABLED   */
00058 /*#define HAL_WWDG_MODULE_ENABLED   */
00059 /*#define HAL_PCD_MODULE_ENABLED   */
00060 #define HAL_CORTEX_MODULE_ENABLED
00061 #define HAL_DMA_MODULE_ENABLED
00062 #define HAL_FLASH_MODULE_ENABLED
00063 #define HAL_GPIO_MODULE_ENABLED
00064 #define HAL_EXTI_MODULE_ENABLED
00065 #define HAL_PWR_MODULE_ENABLED
00066 #define HAL_RCC_MODULE_ENABLED
00067 #define HAL_I2C_MODULE_ENABLED
00068
00069 /* ########################## HSE/HSI Values adaptation ##################### */
00075 #if !defined  (HSE_VALUE)
00076   #define HSE_VALUE    ((uint32_t)8000000)
00077 #endif /* HSE_VALUE */
00078
00083 #if !defined  (HSE_STARTUP_TIMEOUT)
00084   #define HSE_STARTUP_TIMEOUT    ((uint32_t)100)
00085 #endif /* HSE_STARTUP_TIMEOUT */
00086
00092 #if !defined  (HSI_VALUE)
00093   #define HSI_VALUE    ((uint32_t)8000000)
00094 #endif /* HSI_VALUE */
00095
00100 #if !defined  (HSI_STARTUP_TIMEOUT)
00101  #define HSI_STARTUP_TIMEOUT   ((uint32_t)5000)
00102 #endif /* HSI_STARTUP_TIMEOUT */
00103
00107 #if !defined  (HSI14_VALUE)
00108 #define HSI14_VALUE ((uint32_t)14000000)
00111 #endif /* HSI14_VALUE */
00112
00116 #if !defined  (HSI48_VALUE)
00117   #define HSI48_VALUE ((uint32_t)48000000)
00120 #endif /* HSI48_VALUE */
00121
00125 #if !defined  (LSI_VALUE)
00126   #define LSI_VALUE  ((uint32_t)40000)
00127 #endif /* LSI_VALUE */
00133 #if !defined  (LSE_VALUE)
00134   #define LSE_VALUE  ((uint32_t)32768)
00135 #endif /* LSE_VALUE */
00136
00140 #if !defined  (LSE_STARTUP_TIMEOUT)
00141   #define LSE_STARTUP_TIMEOUT    ((uint32_t)5000)
00142 #endif /* LSE_STARTUP_TIMEOUT */
00143
00144 /* Tip: To avoid modifying this file each time you need to use different HSE,
00145    ===  you can define the HSE value in your toolchain compiler preprocessor. */
00146
00147 /* ########################### System Configuration ######################### */
00151 #define  VDD_VALUE                    ((uint32_t)3300)
00152 #define  TICK_INT_PRIORITY            ((uint32_t)3)
00153                                                                              /*  Warning: Must be set
    to higher priority for HAL_Delay()  */
00154                                                                              /*  and HAL_GetTick()
    usage under interrupt context          */
00155 #define  USE_RTOS                     0
00156 #define  PREFETCH_ENABLE              1
00157 #define  INSTRUCTION_CACHE_ENABLE     0
00158 #define  DATA_CACHE_ENABLE            0
00159 #define  USE_SPI_CRC                     0U
00160
00161 #define  USE_HAL_ADC_REGISTER_CALLBACKS         0U /* ADC register callback disabled       */
00162 #define  USE_HAL_CAN_REGISTER_CALLBACKS         0U /* CAN register callback disabled       */
00163 #define  USE_HAL_COMP_REGISTER_CALLBACKS        0U /* COMP register callback disabled      */
00164 #define  USE_HAL_CEC_REGISTER_CALLBACKS         0U /* CEC register callback disabled       */
00165 #define  USE_HAL_DAC_REGISTER_CALLBACKS         0U /* DAC register callback disabled       */
00166 #define  USE_HAL_I2C_REGISTER_CALLBACKS         0U /* I2C register callback disabled       */
00167 #define  USE_HAL_SMBUS_REGISTER_CALLBACKS       0U /* SMBUS register callback disabled     */
00168 #define  USE_HAL_UART_REGISTER_CALLBACKS        0U /* UART register callback disabled      */
00169 #define  USE_HAL_USART_REGISTER_CALLBACKS       0U /* USART register callback disabled     */
00170 #define  USE_HAL_IRDA_REGISTER_CALLBACKS        0U /* IRDA register callback disabled      */
00171 #define  USE_HAL_SMARTCARD_REGISTER_CALLBACKS   0U /* SMARTCARD register callback disabled */
00172 #define  USE_HAL_WWDG_REGISTER_CALLBACKS        0U /* WWDG register callback disabled      */
00173 #define  USE_HAL_RTC_REGISTER_CALLBACKS         0U /* RTC register callback disabled       */
00174 #define  USE_HAL_SPI_REGISTER_CALLBACKS         0U /* SPI register callback disabled       */
00175 #define  USE_HAL_I2S_REGISTER_CALLBACKS         0U /* I2S register callback disabled       */
00176 #define  USE_HAL_TIM_REGISTER_CALLBACKS         0U /* TIM register callback disabled       */
00177 #define  USE_HAL_TSC_REGISTER_CALLBACKS         0U /* TSC register callback disabled       */
00178 #define  USE_HAL_PCD_REGISTER_CALLBACKS         0U /* PCD register callback disabled       */
00179
00180 /* ########################## Assert Selection ############################## */
00185 /* #define USE_FULL_ASSERT   1U */
00186
```

```
00187 /* Includes ------------------------------------------------------------------*/
00192 #ifdef HAL_RCC_MODULE_ENABLED
00193  #include "stm32f0xx_hal_rcc.h"
00194 #endif /* HAL_RCC_MODULE_ENABLED */
00195
00196 #ifdef HAL_GPIO_MODULE_ENABLED
00197  #include "stm32f0xx_hal_gpio.h"
00198 #endif /* HAL_GPIO_MODULE_ENABLED */
00199
00200 #ifdef HAL_EXTI_MODULE_ENABLED
00201   #include "stm32f0xx_hal_exti.h"
00202 #endif /* HAL_EXTI_MODULE_ENABLED */
00203
00204 #ifdef HAL_DMA_MODULE_ENABLED
00205   #include "stm32f0xx_hal_dma.h"
00206 #endif /* HAL_DMA_MODULE_ENABLED */
00207
00208 #ifdef HAL_CORTEX_MODULE_ENABLED
00209  #include "stm32f0xx_hal_cortex.h"
00210 #endif /* HAL_CORTEX_MODULE_ENABLED */
00211
00212 #ifdef HAL_ADC_MODULE_ENABLED
00213  #include "stm32f0xx_hal_adc.h"
00214 #endif /* HAL_ADC_MODULE_ENABLED */
00215
00216 #ifdef HAL_CAN_MODULE_ENABLED
00217  #include "stm32f0xx_hal_can.h"
00218 #endif /* HAL_CAN_MODULE_ENABLED */
00219
00220 #ifdef HAL_CEC_MODULE_ENABLED
00221  #include "stm32f0xx_hal_cec.h"
00222 #endif /* HAL_CEC_MODULE_ENABLED */
00223
00224 #ifdef HAL_COMP_MODULE_ENABLED
00225  #include "stm32f0xx_hal_comp.h"
00226 #endif /* HAL_COMP_MODULE_ENABLED */
00227
00228 #ifdef HAL_CRC_MODULE_ENABLED
00229  #include "stm32f0xx_hal_crc.h"
00230 #endif /* HAL_CRC_MODULE_ENABLED */
00231
00232 #ifdef HAL_DAC_MODULE_ENABLED
00233  #include "stm32f0xx_hal_dac.h"
00234 #endif /* HAL_DAC_MODULE_ENABLED */
00235
00236 #ifdef HAL_FLASH_MODULE_ENABLED
00237  #include "stm32f0xx_hal_flash.h"
00238 #endif /* HAL_FLASH_MODULE_ENABLED */
00239
00240 #ifdef HAL_I2C_MODULE_ENABLED
00241  #include "stm32f0xx_hal_i2c.h"
00242 #endif /* HAL_I2C_MODULE_ENABLED */
00243
00244 #ifdef HAL_I2S_MODULE_ENABLED
00245  #include "stm32f0xx_hal_i2s.h"
00246 #endif /* HAL_I2S_MODULE_ENABLED */
00247
00248 #ifdef HAL_IRDA_MODULE_ENABLED
00249  #include "stm32f0xx_hal_irda.h"
00250 #endif /* HAL_IRDA_MODULE_ENABLED */
00251
00252 #ifdef HAL_IWDG_MODULE_ENABLED
00253  #include "stm32f0xx_hal_iwdg.h"
00254 #endif /* HAL_IWDG_MODULE_ENABLED */
00255
00256 #ifdef HAL_PCD_MODULE_ENABLED
00257  #include "stm32f0xx_hal_pcd.h"
00258 #endif /* HAL_PCD_MODULE_ENABLED */
00259
00260 #ifdef HAL_PWR_MODULE_ENABLED
00261  #include "stm32f0xx_hal_pwr.h"
00262 #endif /* HAL_PWR_MODULE_ENABLED */
00263
00264 #ifdef HAL_RTC_MODULE_ENABLED
00265  #include "stm32f0xx_hal_rtc.h"
00266 #endif /* HAL_RTC_MODULE_ENABLED */
00267
00268 #ifdef HAL_SMARTCARD_MODULE_ENABLED
00269  #include "stm32f0xx_hal_smartcard.h"
00270 #endif /* HAL_SMARTCARD_MODULE_ENABLED */
00271
00272 #ifdef HAL_SMBUS_MODULE_ENABLED
00273  #include "stm32f0xx_hal_smbus.h"
00274 #endif /* HAL_SMBUS_MODULE_ENABLED */
00275
00276 #ifdef HAL_SPI_MODULE_ENABLED
00277  #include "stm32f0xx_hal_spi.h"
```

```
00278 #endif /* HAL_SPI_MODULE_ENABLED */
00279
00280 #ifdef HAL_TIM_MODULE_ENABLED
00281  #include "stm32f0xx_hal_tim.h"
00282 #endif /* HAL_TIM_MODULE_ENABLED */
00283
00284 #ifdef HAL_TSC_MODULE_ENABLED
00285  #include "stm32f0xx_hal_tsc.h"
00286 #endif /* HAL_TSC_MODULE_ENABLED */
00287
00288 #ifdef HAL_UART_MODULE_ENABLED
00289  #include "stm32f0xx_hal_uart.h"
00290 #endif /* HAL_UART_MODULE_ENABLED */
00291
00292 #ifdef HAL_USART_MODULE_ENABLED
00293  #include "stm32f0xx_hal_usart.h"
00294 #endif /* HAL_USART_MODULE_ENABLED */
00295
00296 #ifdef HAL_WWDG_MODULE_ENABLED
00297  #include "stm32f0xx_hal_wwdg.h"
00298 #endif /* HAL_WWDG_MODULE_ENABLED */
00299
00300 /* Exported macro ------------------------------------------------------------*/
00301 #ifdef  USE_FULL_ASSERT
00310   #define assert_param(expr) ((expr) ? (void)0U : assert_failed((uint8_t *)__FILE__, __LINE__))
00311 /* Exported functions ------------------------------------------------------- */
00312   void assert_failed(uint8_t* file, uint32_t line);
00313 #else
00314   #define assert_param(expr) ((void)0U)
00315 #endif /* USE_FULL_ASSERT */
00316
00317 #ifdef __cplusplus
00318 }
00319 #endif
00320
00321 #endif /* __STM32F0xx_HAL_CONF_H */
00322
```

9.29 SENSEHAT INTEGRATION/Core/Inc/stm32f0xx_it.h File Reference

This file contains the headers of the interrupt handlers.

Functions

- void **NMI_Handler** (void)

  This function handles Non maskable interrupt.
- void **HardFault_Handler** (void)

  This function handles Hard fault interrupt.
- void **SVC_Handler** (void)

  This function handles System service call via SWI instruction.
- void **PendSV_Handler** (void)

  This function handles Pendable request for system service.
- void **SysTick_Handler** (void)

  This function handles System tick timer.
- void **TIM2_IRQHandler** (void)

  This function handles TIM2 global interrupt.

9.29.1 Detailed Description

This file contains the headers of the interrupt handlers.

Attention

9.30 stm32f0xx_it.h

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00018 /* USER CODE END Header */
00019
00020 /* Define to prevent recursive inclusion -------------------------------------*/
00021 #ifndef __STM32F0xx_IT_H
00022 #define __STM32F0xx_IT_H
00023
00024 #ifdef __cplusplus
00025  extern "C" {
00026 #endif
00027
00028 /* Private includes ----------------------------------------------------------*/
00029 /* USER CODE BEGIN Includes */
00030
00031 /* USER CODE END Includes */
00032
00033 /* Exported types ------------------------------------------------------------*/
00034 /* USER CODE BEGIN ET */
00035
00036 /* USER CODE END ET */
00037
00038 /* Exported constants --------------------------------------------------------*/
00039 /* USER CODE BEGIN EC */
00040
00041 /* USER CODE END EC */
00042
00043 /* Exported macro ------------------------------------------------------------*/
00044 /* USER CODE BEGIN EM */
00045
00046 /* USER CODE END EM */
00047
00048 /* Exported functions prototypes ---------------------------------------------*/
00049 void NMI_Handler(void);
00050 void HardFault_Handler(void);
00051 void SVC_Handler(void);
00052 void PendSV_Handler(void);
00053 void SysTick_Handler(void);
00054 void TIM2_IRQHandler(void);
00055 /* USER CODE BEGIN EFP */
00056
00057 /* USER CODE END EFP */
00058
00059 #ifdef __cplusplus
00060 }
00061 #endif
00062
00063 #endif /* __STM32F0xx_IT_H */
```

9.31 tmp102.h

```
00001 /*
00002  * TMP102 Temperature Sensor Library
00003  * Author: Travimadox Webb
00004  * Postion: Embedded Software Engineer
00005  * Company: Imperium LLC
00006  * Date: 6th of May 2023
00007  */
00008
00009 #ifndef TMP102_H
00010 #define TMP102_H
00011
00012 #include "stm32f0xx_hal.h"
00013
00014 // TMP102 I2C address (default: 0x48)
00015 #define TMP102_I2C_ADDRESS 0x48
00016
00017 // TMP102 register addresses
00018 #define TMP102_REG_TEMPERATURE 0x00
00019 #define TMP102_REG_CONFIG 0x01
00020
00021 // TMP102 configuration settings
00022 #define TMP102_CONFIG_CONTINUOUS_CONVERSION 0x0000
00023 #define TMP102_CONFIG_SHUTDOWN_MODE 0x0100
00024
00025 // Initialize the TMP102 sensor
00026 HAL_StatusTypeDef TMP102_Init(I2C_HandleTypeDef *hi2c);
00027
00028 // Read temperature from the TMP102 sensor
```

```
00029 float TMP102_ReadTemperature(I2C_HandleTypeDef *hi2c);
00030
00031 #endif // TMP102_H
```

9.32 tmp102.h

```
00001 /*
00002  * TMP102 Temperature Sensor Library
00003  * Author: Travimadox Webb
00004  * Postion: Embedded Software Engineer
00005  * Company: Imperium LLC
00006  * Date: 6th of May 2023
00007  */
00008
00009 #ifndef TMP102_H
00010 #define TMP102_H
00011
00012 #include "stm32f0xx_hal.h"
00013
00014 // TMP102 I2C address (default: 0x48)
00015 #define TMP102_I2C_ADDRESS 0x48
00016
00017 // TMP102 register addresses
00018 #define TMP102_REG_TEMPERATURE 0x00
00019 #define TMP102_REG_CONFIG 0x01
00020
00021 // TMP102 configuration settings
00022 #define TMP102_CONFIG_CONTINUOUS_CONVERSION 0x0000
00023 #define TMP102_CONFIG_SHUTDOWN_MODE 0x0100
00024
00025 // Initialize the TMP102 sensor
00026 HAL_StatusTypeDef TMP102_Init(I2C_HandleTypeDef *hi2c);
00027
00028 // Read temperature from the TMP102 sensor
00029 float TMP102_ReadTemperature(I2C_HandleTypeDef *hi2c);
00030
00031 #endif // TMP102_H
```

9.33 SENSEHAT INTEGRATION/Core/Src/main.c File Reference

: Main program body

```
#include "main.h"
#include "rtc.h"
#include "ldr.h"
#include "tmp102.h"
#include "EEPROM.h"
#include "string.h"
#include <math.h>
#include <stdbool.h>
```

Functions

- void SystemClock_Config (void)

  System Clock Configuration.
- int main (void)

  The application entry point.
- void read_and_store_data (void)

  Read and store data.
- void read_and_transmit_all_data (void)

Read and transmit all data This function reads all the data stored in the EEPROM memory and transmits it over UART. It loops through all the pages of the EEPROM, reads the data from each page, and transmits it using UART communication. There is a delay between each UART transmission to ensure proper data transmission.

- void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)

  GPIO EXTI callback function This function is called when an EXTI interrupt event is triggered on a GPIO pin. It specifically handles the interrupt triggered by the PB10 pin, indicating that the USB has been plugged in. When this interrupt occurs, the function starts transmitting all the stored data over UART.
- void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim)

  TIM period elapsed callback function.
- void Error_Handler (void)

  This function is executed in case of error occurrence.

Variables

- ADC_HandleTypeDef **hadc**
- I2C_HandleTypeDef **hi2c1**
- RTC_HandleTypeDef **hrtc**
- TIM_HandleTypeDef **htim2**
- UART_HandleTypeDef **huart1**
- uint8_t **hours** = 07
- uint8_t **minutes** = 16
- uint8_t **seconds** = 0
- uint8_t **day** = 12
- uint8_t **month** = 5
- uint8_t **year** = 23
- uint8_t **read_hours**
- uint8_t **read_minutes**
- uint8_t **read_seconds**
- char **time_string** [9]
- uint8_t **dayread**
- uint8_t **monthread**
- uint8_t **yearread**
- char **date_string** [11]
- float **light_intensity**
- float **temperature**
- char **temp_string** [10]
- char **light_string** [10]
- HAL_StatusTypeDef **status**
- uint16_t **current_page** = 0
- bool **eeprom_full** = false
- bool **usb_plugged** = false

9.33.1 Detailed Description

: Main program body

Attention

Author

Rumbidzai Mashumba, Phomolo Makina, Travimadox Webb @company Imperium LLC

Date

13th of May 2023

9.33.2 Function Documentation

9.33.2.1 Error_Handler()

```
void Error_Handler (
            void  )
```

This function is executed in case of error occurrence.

Return values

None	

9.33.2.2 HAL_GPIO_EXTI_Callback()

```
void HAL_GPIO_EXTI_Callback (
            uint16_t GPIO_Pin )
```

GPIO EXTI callback function This function is called when an EXTI interrupt event is triggered on a GPIO pin. It specifically handles the interrupt triggered by the PB10 pin, indicating that the USB has been plugged in. When this interrupt occurs, the function starts transmitting all the stored data over UART.

Parameters

GPIO_Pin	The GPIO pin that triggered the interrupt

9.33.2.3 HAL_TIM_PeriodElapsedCallback()

```
void HAL_TIM_PeriodElapsedCallback (
            TIM_HandleTypeDef * htim )
```

TIM period elapsed callback function.

This function is called when the period of a TIM (Timer) peripheral has elapsed. It specifically handles the callback for the timer used to track the elapsed time of 60 seconds. When this callback is triggered, the function reads data from the sensors and stores it.

9.33.2.4 main()

```
int main (
             void  )
```

The application entry point.

Return values

int	

9.33.2.5 read_and_store_data()

```
void read_and_store_data (
             void  )
```

Read and store data.

This function reads the light intensity, temperature, date, and time, and stores the data in the EEPROM memory.

The data is formatted using CSV (Comma-Separated Values) format and written to the EEPROM memory.

The function checks if the EEPROM is full before performing any write operations.

Note

If the EEPROM is full, the function does not perform any write operations.

9.33.2.6 read_and_transmit_all_data()

```
void read_and_transmit_all_data (
             void  )
```

Read and transmit all data This function reads all the data stored in the EEPROM memory and transmits it over UART. It loops through all the pages of the EEPROM, reads the data from each page, and transmits it using UART communication. There is a delay between each UART transmission to ensure proper data transmission.

Note

This function assumes that the EEPROM has been filled with data and the eeprom_full flag is not checked. If the EEPROM is not filled with data, this function may transmit garbage values.

9.33.2.7 SystemClock_Config()

```
void SystemClock_Config (
             void  )
```

System Clock Configuration.

Return values

None	

Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef structure.

Initializes the CPU, AHB and APB buses clocks

9.34 SENSEHAT INTEGRATION/Core/Src/stm32f0xx_hal_msp.c File Reference

This file provides code for the MSP Initialization and de-Initialization codes.

```
#include "main.h"
```

Functions

- void HAL_MspInit (void)
- void HAL_ADC_MspInit (ADC_HandleTypeDef ∗hadc)

  ADC MSP Initialization This function configures the hardware resources used in this example.
- void HAL_ADC_MspDeInit (ADC_HandleTypeDef ∗hadc)

  ADC MSP De-Initialization This function freeze the hardware resources used in this example.
- void HAL_I2C_MspInit (I2C_HandleTypeDef ∗hi2c)

  I2C MSP Initialization This function configures the hardware resources used in this example.
- void HAL_I2C_MspDeInit (I2C_HandleTypeDef ∗hi2c)

  I2C MSP De-Initialization This function freeze the hardware resources used in this example.
- void HAL_RTC_MspInit (RTC_HandleTypeDef ∗hrtc)

  RTC MSP Initialization This function configures the hardware resources used in this example.
- void HAL_RTC_MspDeInit (RTC_HandleTypeDef ∗hrtc)

  RTC MSP De-Initialization This function freeze the hardware resources used in this example.
- void HAL_TIM_Base_MspInit (TIM_HandleTypeDef ∗htim_base)

  TIM_Base MSP Initialization This function configures the hardware resources used in this example.
- void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef ∗htim_base)

  TIM_Base MSP De-Initialization This function freeze the hardware resources used in this example.
- void HAL_UART_MspInit (UART_HandleTypeDef ∗huart)

  UART MSP Initialization This function configures the hardware resources used in this example.
- void HAL_UART_MspDeInit (UART_HandleTypeDef ∗huart)

  UART MSP De-Initialization This function freeze the hardware resources used in this example.

9.34.1 Detailed Description

This file provides code for the MSP Initialization and de-Initialization codes.

Attention

9.34.2 Function Documentation

9.34.2.1 HAL_ADC_MspDeInit()

```
void HAL_ADC_MspDeInit (
            ADC_HandleTypeDef * hadc )
```

ADC MSP De-Initialization This function freeze the hardware resources used in this example.

Parameters

hadc	ADC handle pointer

Return values

None	

ADC GPIO Configuration PA5 ---—> ADC_IN5

9.34.2.2 HAL_ADC_MspInit()

```
void HAL_ADC_MspInit (
            ADC_HandleTypeDef * hadc )
```

ADC MSP Initialization This function configures the hardware resources used in this example.

Parameters

hadc	ADC handle pointer

Return values

None	

ADC GPIO Configuration PA5 ---—> ADC_IN5

9.34.2.3 HAL_I2C_MspDeInit()

```
void HAL_I2C_MspDeInit (
            I2C_HandleTypeDef * hi2c )
```

I2C MSP De-Initialization This function freeze the hardware resources used in this example.

Parameters

hi2c	I2C handle pointer

Return values

None	

I2C1 GPIO Configuration PB6 ------> I2C1_SCL PB7 ------> I2C1_SDA

9.34.2.4 HAL_I2C_MspInit()

```
void HAL_I2C_MspInit (
            I2C_HandleTypeDef * hi2c )
```

I2C MSP Initialization This function configures the hardware resources used in this example.

Parameters

hi2c	I2C handle pointer

Return values

None	

I2C1 GPIO Configuration PB6 ------> I2C1_SCL PB7 ------> I2C1_SDA

9.34.2.5 HAL_MspInit()

```
void HAL_MspInit (
            void  )
```

Initializes the Global MSP.

9.34.2.6 HAL_RTC_MspDeInit()

```
void HAL_RTC_MspDeInit (
            RTC_HandleTypeDef * hrtc )
```

RTC MSP De-Initialization This function freeze the hardware resources used in this example.

Parameters

hrtc	RTC handle pointer

Return values

None	

9.34.2.7 HAL_RTC_MspInit()

```
void HAL_RTC_MspInit (
            RTC_HandleTypeDef * hrtc )
```

RTC MSP Initialization This function configures the hardware resources used in this example.

Parameters

hrtc	RTC handle pointer

Return values

None	

9.34.2.8 HAL_TIM_Base_MspDeInit()

```
void HAL_TIM_Base_MspDeInit (
            TIM_HandleTypeDef * htim_base )
```

TIM_Base MSP De-Initialization This function freeze the hardware resources used in this example.

Parameters

htim_base	TIM_Base handle pointer

Return values

None	

9.34.2.9 HAL_TIM_Base_MspInit()

```
void HAL_TIM_Base_MspInit (
            TIM_HandleTypeDef * htim_base )
```

TIM_Base MSP Initialization This function configures the hardware resources used in this example.

Parameters

htim_base	TIM_Base handle pointer

Return values

None	

9.34.2.10 HAL_UART_MspDeInit()

```
void HAL_UART_MspDeInit (
            UART_HandleTypeDef * huart )
```

UART MSP De-Initialization This function freeze the hardware resources used in this example.

Parameters

huart	UART handle pointer

Return values

None	

USART1 GPIO Configuration PA9 ------> USART1_TX PA10 ------> USART1_RX

9.34.2.11 HAL_UART_MspInit()

```
void HAL_UART_MspInit (
            UART_HandleTypeDef * huart )
```

UART MSP Initialization This function configures the hardware resources used in this example.

Parameters

huart	UART handle pointer

Return values

None	

USART1 GPIO Configuration PA9 ------> USART1_TX PA10 ------> USART1_RX

9.35 SENSEHAT INTEGRATION/Core/Src/stm32f0xx_it.c File Reference

Interrupt Service Routines.

```
#include "main.h"
#include "stm32f0xx_it.h"
```

Functions

- void **NMI_Handler** (void)

  This function handles Non maskable interrupt.
- void **HardFault_Handler** (void)

  This function handles Hard fault interrupt.
- void **SVC_Handler** (void)

  This function handles System service call via SWI instruction.
- void **PendSV_Handler** (void)

  This function handles Pendable request for system service.
- void **SysTick_Handler** (void)

  This function handles System tick timer.
- void **TIM2_IRQHandler** (void)

  This function handles TIM2 global interrupt.

Variables

- TIM_HandleTypeDef **htim2**

9.35.1 Detailed Description

Interrupt Service Routines.

Attention

9.36 SENSEHAT INTEGRATION/Core/Src/syscalls.c File Reference

STM32CubeIDE Minimal System calls file.

```
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <signal.h>
#include <time.h>
#include <sys/time.h>
#include <sys/times.h>
```

Functions

- int **__io_putchar** (int ch) __attribute__((weak))
- int **__io_getchar** (void)
- void **initialise_monitor_handles** ()
- int **_getpid** (void)
- int **_kill** (int pid, int sig)
- void **_exit** (int status)
- **__attribute__** ((weak))
- int **_close** (int file)
- int **_fstat** (int file, struct stat *st)
- int **_isatty** (int file)
- int **_lseek** (int file, int ptr, int dir)
- int **_open** (char *path, int flags,...)
- int **_wait** (int *status)
- int **_unlink** (char *name)
- int **_times** (struct tms *buf)
- int **_stat** (char *file, struct stat *st)
- int **_link** (char *old, char *new)
- int **_fork** (void)
- int **_execve** (char *name, char **argv, char **env)

Variables

- char ** **environ** = __env

9.36.1 Detailed Description

STM32CubeIDE Minimal System calls file.

Author

Auto-generated by STM32CubeIDE

```
For more information about which c-functions
need which of these lowlevel functions
please consult the Newlib libc-manual
```

Attention

9.37 SENSEHAT INTEGRATION/Core/Src/sysmem.c File Reference

STM32CubeIDE System Memory calls file.

```
#include <errno.h>
#include <stdint.h>
```

Functions

- void * _sbrk (ptrdiff_t incr)

  _sbrk() allocates memory to the newlib heap and is used by malloc and others from the C library

9.37.1 Detailed Description

STM32CubeIDE System Memory calls file.

Author

Generated by STM32CubeIDE

```
For more information about which C functions
need which of these lowlevel functions
please consult the newlib libc manual
```

Attention

9.37.2 Function Documentation

9.37.2.1 _sbrk()

```
void * _sbrk (
            ptrdiff_t incr )
```

_sbrk() allocates memory to the newlib heap and is used by malloc and others from the C library

```
* ##########################################################################
* #  .data  #  .bss  #       newlib heap       #          MSP stack          #
* #         #        #                         # Reserved by _Min_Stack_Size #
* ##########################################################################
* ^-- RAM start      ^-- _end                             _estack, RAM end --^
*
```

This implementation starts allocating at the '_end' linker symbol The '_Min_Stack_Size' linker symbol reserves a memory for the MSP stack The implementation considers '_estack' linker symbol to be RAM end NOTE: If the MSP stack, at any point during execution, grows larger than the reserved size, please increase the '_Min_Stack_Size'.

Parameters

incr	Memory size

Returns

Pointer to allocated memory

9.38 SENSEHAT INTEGRATION/Core/Src/system_stm32f0xx.c File Reference

CMSIS Cortex-M0 Device Peripheral Access Layer System Source File.

```
#include "stm32f0xx.h"
```

Macros

- #define HSE_VALUE ((uint32_t)8000000)
- #define HSI_VALUE ((uint32_t)8000000)
- #define HSI48_VALUE ((uint32_t)48000000)

Functions

- void SystemInit (void)

  Setup the microcontroller system.
- void SystemCoreClockUpdate (void)

  Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

Variables

- uint32_t **SystemCoreClock** = 8000000
- const uint8_t **AHBPrescTable** [16] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}
- const uint8_t **APBPrescTable** [8] = {0, 0, 0, 0, 1, 2, 3, 4}

9.38.1 Detailed Description

CMSIS Cortex-M0 Device Peripheral Access Layer System Source File.

Author

MCD Application Team

1. This file provides two functions and one global variable to be called from user application:
   - SystemInit(): This function is called at startup just after reset and before branch to main program. This call is made inside the "startup_stm32f0xx.s" file.
   - SystemCoreClock variable: Contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.
   - SystemCoreClockUpdate(): Updates the variable SystemCoreClock and must be called whenever the core clock is changed during program execution.

Attention

9.39 SENSEHAT INTEGRATION/Core/Src/tmp102.c File Reference

TMP102 Temperature Sensor Library.

```
#include "tmp102.h"
#include <math.h>
```

Functions

- HAL_StatusTypeDef TMP102_Init (I2C_HandleTypeDef *hi2c)

  Initialize the TMP102 temperature sensor.
- float TMP102_ReadTemperature (I2C_HandleTypeDef *hi2c)

  Read temperature from the TMP102 sensor.

9.39.1 Detailed Description

TMP102 Temperature Sensor Library.

Author

Travimadox Webb @position Embedded Software Engineer @company Imperium LLC

Date

6th of May 2023

9.39.2 Function Documentation

9.39.2.1 TMP102_Init()

```
HAL_StatusTypeDef TMP102_Init (
            I2C_HandleTypeDef * hi2c )
```

Initialize the TMP102 temperature sensor.

Parameters

hi2c	Pointer to an I2C_HandleTypeDef structure that contains the configuration information for the specified I2C peripheral.

Return values

HAL	status

9.39.2.2 TMP102_ReadTemperature()

```
float TMP102_ReadTemperature (
            I2C_HandleTypeDef * hi2c )
```

Read temperature from the TMP102 sensor.

Parameters

hi2c	Pointer to an I2C_HandleTypeDef structure that contains the configuration information for the specified I2C peripheral.

Returns

Temperature in degrees Celsius as a float.

9.40 TMP102/tmp102.c File Reference

TMP102 Temperature Sensor Library.

```
#include "tmp102.h"
#include <math.h>
```

Functions

- HAL_StatusTypeDef TMP102_Init (I2C_HandleTypeDef *hi2c)

  Initialize the TMP102 temperature sensor.
- float TMP102_ReadTemperature (I2C_HandleTypeDef *hi2c)

  Read temperature from the TMP102 sensor.

9.40.1 Detailed Description

TMP102 Temperature Sensor Library.

Author

Travimadox Webb @position Embedded Software Engineer @company Imperium LLC

Date

6th of May 2023

9.40.2 Function Documentation

9.40.2.1 TMP102_Init()

```
HAL_StatusTypeDef TMP102_Init (
            I2C_HandleTypeDef * hi2c )
```

Initialize the TMP102 temperature sensor.

Parameters

hi2c	Pointer to an I2C_HandleTypeDef structure that contains the configuration information for the specified I2C peripheral.

Return values

HAL	status

9.40.2.2 TMP102_ReadTemperature()

```
float TMP102_ReadTemperature (
            I2C_HandleTypeDef * hi2c )
```

Read temperature from the TMP102 sensor.

Parameters

hi2c	Pointer to an I2C_HandleTypeDef structure that contains the configuration information for the specified I2C peripheral.

Returns

Temperature in degrees Celsius as a float.

Index