Proximity and Temperature HAT: Software Documentation

Contents

Overview

This document contains information on all the code that has been written and compiled as a companion to the proximity and temperature HAT that was designed. It also contains all files needed to run the HAT as intended.

Files

User Application Files

The basic software of the Proximity and Temperature HAT interfacing with the STM32f0 discovery board is comprised of the following files. This have been written or compiled by the engineering team.

File	Description
main.h/ .c	This file contains all of the functions written to allow a more user-friendly interface with the other files. It also initialises the STM32f0. User code can be written here to change the functioning of the board and microcontroller.
EEPROM.h/ .c	This file is a library written by **controllerstech** downloadable at https://github.com/controllerstech/STM32/tree/master/EEPROM_STM32.

	This library was used to simplify the user commands needed to read, write to the EEPROM and erase the EEPROM
prox.h/ .c	Contains definitions for functions used to control the proximity sensor
temp.h/ .c	Contains definitions for functions used to control the temperature sensor
types.h	Contains the definitions for types used throughout the API
sensors.h	Contains the public facing functions of the sensor api and is intended to be used by the end user

Table 1: User Application Files

HAL files

The next files are HAL drivers needed to run the above code. The descriptions have been obtained from the HAL API documentation found at https://www.st.com/resource/en/user_manual/dm00105879-description-of-stm32f4-hal-and-ll-drivers-stmicroelectronics.pdf.

File	Description
stm32f0xx_hal_conf.h	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.
stm32f0xx_it.h/ .c	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in stm32f4xx_hal.c) used as HAL timebase. By default, this function is called each 1ms in Systick ISR. The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
stm32f0xx_hal_msp.c	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
system_stmf0xx.c	This file contains SystemInit() that is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library). This is to be done using the HAL APIs in the user files. It allows relocating the vector table in internal SRAM and configuring the FSMC/FMC (when available) to use the external SRAM or SDRAM mounted on the evaluation board as data memory.

Table 2: HAL Files

Functions

Main.c

SystemClock_Config

Function Name

void SystemClock_Config(void);

Function Description

Initializes the system clock for the STM32f0

Parameters

None

Return Values

None

MX_GPIO_Init

Function Name

static void MX_GPIO_Init(void);

Function Description

Initializes the GPIO for the STM32f0 including the pins for the USB detect (including the associated interrupt), Temperature active, Proximity active, Proximity Interrupt and discovery board LEDS.

Parameters

None

Return Values

None

MX_I2C1_Init

Function Name

static void MX_I2C1_Init(void);

Function Description

Initializes the I2C1 communication line for the STM32f0 using pins PB7 and PB8.

Parameters

None

Return Values

None

MX_RTC_Init

Function Name

static void MX_RTC_Init(void);

Function Description

Initializes the RTC (real time clock) onboard the STM32f0. Default time is 16:30:00 on 19/05/2022

Parameters

None

Return Values

None

MX_ADC_Init

Function Name

static void MX_ADC_Init(void);

Function Description

Initializes the ADC for the STM32f0 to receive data from the temperature sensor

Parameters

None

Return Values

None

set_time

Function Name

void set_time(uint8_t hour,uint8_t minute, uint8_t second, uint8_t Month, uint8_t Date, uint8_t Year);

Function Description

Sets the time of the real time clock to a user specified time

Parameters

Hour: user defined hour
minute: user defined minute
second: user defined second
Month: user defined month. Can use RTC definitions eg. RTC_MONTH_MAY
Date: user defined date
Year: user difined year. Needs to be less then 255 so store year-2000 and add the 2000 when displaying later

Return Values

None

get_time_to_store

Function Name

void get_time_to_store(uint8_t (*time)[3],uint8_t (*date)[3]);

Function Description

Retrives the current time and date from the RTC and stores it in 2 arrays of length 3 and size one byte. One array has time and the other has the date

Parameters

time: pointer to user defined array of 3 variables that will store current time
date: pointer to user defined array of 3 variables that will store current time

Return Values

None

store_one_set

Function Name

void store_one_set(uint32_t *temp, uint32_t *proximity);

Function Description

Stores one set of data in the EEPROM which includes the ID of that set, the temperature data, proximity data and the time from the RTC.

Parameters

temp: pointer to user defined uint32 containing the temperature
proximity: pointer to user defined uint32 containing the proximity

Return Values

None

read_all_data

Function Name

void read_all_data (uint8_t (*dataRead)[32678]);

Function Description

Reads all of the data currently stored on the EEPROM and stores it in an array

Parameters

dataRead: pointer to user defined array that will store the EEPROM data

Return Values

None

get_sensor_data

Function Name

void get_sensor_data (uint32_t *proximity,uint32_t *temp);

Function Description

Stores the data currently received from the 2 sensors in user defined variables

Parameters

proximity: pointer to user defined byte to contain proximity
temp: pointer to user defined byte to contain proximity

Return Values

None

get_sensor_data

Function Name

void Intialise(void);

Function Description

Initialises all of the STM32 functions as well as starting the sensors. It contains all previously mentioned initialisation functions

Parameters

None

Return Values

None

HAL_GPIO_EXTI_Callback

Function Name

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin);

Function Description

Defines the action taken of the USB detect interrupt

Parameters

GPIO_Pin: Pin connected to the interrupt

Return Values

None

Global Variables

uint8_t ID – stores the current value of the ID that is used to store values in the EEPROM

EEPROM.c

All of the following references and information have been copied from the original library

EEPROM_Write

Function Name

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

Function Description

write the data to the EEPROM

Parameters

page: the number of the start page. Range from 0 to PAGE_NUM-1
offset: the start byte offset in the page. Range from 0 to PAGE_SIZE-1
data: the pointer to the data to write in bytes
size: the size of the data

Return Values

None

EEPROM_Write_NUM

Function Name

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

Function Description

Write the Float/Integer values to the EEPROM

Parameters

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

Return Values

None

EEPROM_Read_NUM

Function Name

float EEPROM_Read_NUM (uint16_t page, uint16_t offset)

Function Description

Reads the single Float/Integer values from the EEPROM

Parameters

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

Return Values

the float/integer value

EEPROM_Read

Function Name

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

Function Description

READ the data from the EEPROM

Parameters

page: the number of the start page. Range from 0 to PAGE_NUM-1
offset: the start byte offset in the page. Range from 0 to PAGE_SIZE-1
data: the pointer to the data to write in bytes
size: the size of the data

Return Values

None

EEPROM_PageErase

Function Name

float EEPROM_Read_NUM (uint16_t page, uint16_t offset)

Function Description

Erase a page in the EEPROM Memory. In order to erase multiple pages, just use this function in the for loop

Parameters

page: the number of the pages to erase

Return Values

None

Sensors.h

The following list of functions comprise the public facing functions of the sensor api and is intended to be used by the end user

Sensors_Start

Function Name

```
SensorErrorType Sensors_Start(ADC_HandleTypeDef *hadc, I2C_HandleTypeDef* i2cHandler)
```

Function Description

Initialize the Sensors to start the data collection process

Parameters

- **hadc** - the handler of the adc configured by the user. Instructions about specific ports can be found in the readme
- **i2cHandler** - the handler of the i2c configuration defined by the user.

Return Values

SensorErrorType – Indicating the result of the operation

Sensors_Stop

Function Name

```
SensorErrorType Sensors_Stop(void)
```

Function Description

Places both sensors in a standby state

Parameters

N/A

Return Values

SensorErrorType – Indicating the result of the operation

Sensors_GetMeasurement

Function Name

```
SensorErrorType Sensors_GetMeasurement(struct SensorData *out, int maxTime)
```

Function Description

Obtains all measurements and stores it in a pointer to sensor data

Parameters

- out - a pointer to the struct*{SensorData}* in which the results from measurements will be stored. End user is responsible for creating and managing the struct
- maxTime – the maximum time the operation should take, if this time is exceeded, the function will abort with an error

Return Values

SensorErrorType – Indicating the result of the operation

Types.h

Contains the definitions for types used throughout the API

SensorData

Definition

```
struct SensorData {
	float temp;
	int32_t prox;
};
```

Description

This struct will hold a result from a polling measurement operation.

SensorErrorType

Definition

```
typedef enum SensorErrorType;
```

Description

An enum holding the result from various operations in the api. Below are the enum fields as well as their descriptions

- OK – The function was successful
- PROX_CALIBRATION_ERROR – The proximity sensor could not calibrate itself and thus could not start
- PROX_INIT_ERROR – The proximity sensor could not start
- PROX_MEASURE_ERROR – There was an error while getting a measurement
- PROX_NOT_RESPONDING – The proximity sensor did not respond when prompted for a result

The following section describes private definitions which should not be used by the end user

Prox.h

Contains definitions for functions used to control the proximity sensor

ProximitySensor_Start

Function Name

```
SensorErrorType ProximitySensor_Start(I2C_HandleTypeDef* i2cHandler)
```

Function Description

Starts the proximity sensor

Parameters

- i2cHandler - the handler of the i2c configuration defined by the user.

Return Values

SensorErrorType – Indicating the result of the operation

EnsureHandlersValid

Function Name

```
void EnsureHandlersValid(void)
```

Function Description

Internal function to ensure that the various handlers the api uses are still valid

Parameters

N/A

Return Values

N/A

ProximitySensor_GetSingleShotMeasurement

Function Name

```
SensorErrorType ProximitySensor_GetSingleShotMeasurement(int32_t* result)
```

Function Description

Gets a distance measurement once at a time

Parameters

- result - A pointer to the integer variable holding the result of the operation in millimetres

Return Values

SensorErrorType – Indicating the result of the operation

ProximitySensor_Stop

Function Name

```
SensorErrorType ProximitySensor_Stop(void)
```

Function Description

Places the proximity sensor in standby mode

Parameters

N/A

Return Values

SensorErrorType – Indicating the result of the operation

ProximitySensor_IsReady

Function Name

```
SensorErrorType ProximitySensor_IsReady(void)
```

Function Description

Checks whether the proximity sensor is ready and responding

Parameters

N/A

Return Values

SensorErrorType – Indicating the result of the operation

Interrupt based functions for use if required

ProximitySensor_Start_IT

Function Name

```
SensorErrorType ProximitySensor_Start_IT(I2C_HandleTypeDef* i2cHandler, int freq)
```

Function Description

Starts the proximity sensor in interrupt mode. User is responsible for setting up the GPIO interrupt in hardware

Parameters

- i2cHandler - the handler of the i2c configuration defined by the user.
- Freq – The frequency at which interrupts will occur

Return Values

SensorErrorType – Indicating the result of the operation

ProximitySensor_OnInterruptStarted

Function Name

```
SensorErrorType ProximitySensor_OnInterruptStarted(void)
```

Function Description

Function to be called once the interrupt detection has been set up

Parameters

N/A

Return Values

SensorErrorType – Indicating the result of the operation

ProximitySensor_OnInterruptDetected

Function Name

```
SensorErrorType ProximitySensor_OnInterruptDetected(int32_t* result)
```

Function Description

Called when the interrupt fires. Will output the measurement results

Parameters

- result - A pointer to the integer variable holding the result of the operation in millimetres

Return Values

SensorErrorType – Indicating the result of the operation

Temp.h

Contains definitions for functions used to control the temperature sensor

TempSensor_Start

Function Name

```
SensorErrorType TempSensor_Start(ADC_HandleTypeDef *hadc)
```

Function Description

Starts the temperature sensor

Parameters

- **hadc** - the handler of the adc configured by the user. Instructions about specific ports can be found in the readme

Return Values

SensorErrorType – Indicating the result of the operation

TempSensor_GetMeasurement

Function Name

```
SensorErrorType TempSensor_GetMeasurement(int maxTime, float* result)
```

Function Description

Get the current measurement from the temperature sensor

Parameters

- result - is a pointer to a double which will hold the result
- maxTime - is the maximum amount of time you can wait before a result needs to be returned. Can be dictated by the sampling rate

Return Values

SensorErrorType – Indicating the result of the operation

TempSensor_HasStarted

Function Name

```
SensorErrorType TempSensor_HasStarted(void)
```

Function Description

Checks if the temperature has started

Parameters

N/A

Return Values

SensorErrorType – Indicating the result of the operation

TempSensor_Stop

Function Name

```
SensorErrorType TempSensor_Stop(void)
```

Function Description

Stop the temperature sensor

Parameters

N/A

Return Values

SensorErrorType – Indicating the result of the operation

The following files are part of source code taken from the VL6180 Api and thus, documentation can be found at the STMicroelectronics website

- vl6180_api.h/ .c
- vl6180_cfg.h
- vl6180_def.h

- vl6180_i2c.h/ .c
- vl6180_platform.h/ .c