



ZMOD4510 Ultra-Low Power Ozone Firmware Documentation

Firmware Version: 5.0.0

Contents

1	Overview	1
1.1	Folder Structure	2
2	Module Index	3
2.1	Modules	3
3	Data Structure Index	4
3.1	Data Structures	4
4	File Index	5
4.1	File List	5
5	Module Documentation	6
5.1	Ultra-Low Power Ozone Library API	6
5.2	ZMOD4xxx Sensor API	7
5.2.1	Detailed Description	8
5.2.2	Data Structure Documentation	9
5.2.2.1	struct zmod4xxx_conf_str	9
5.2.2.2	struct zmod4xxx_conf	9
5.2.2.3	struct zmod4xxx_dev_t	9
5.2.3	Macro Definition Documentation	11
5.2.3.1	STATUS_ACCESS_CONFLICT_MASK	11
5.2.3.2	STATUS_ALARM_MASK	11
5.2.3.3	STATUS_LAST_SEQ_STEP_MASK	11

5.2.3.4	STATUS_POR_EVENT_MASK	11
5.2.3.5	STATUS_SEQUENCER_RUNNING_MASK	12
5.2.3.6	STATUS_SLEEP_TIMER_ENABLED_MASK	12
5.2.4	Typedef Documentation	12
5.2.4.1	zmod4xxx_delay_ptr_p	12
5.2.4.2	zmod4xxx_i2c_ptr_t	12
5.2.5	Enumeration Type Documentation	13
5.2.5.1	zmod4xxx_err	13
5.2.6	Function Documentation	13
5.2.6.1	zmod4xxx_calc_factor()	13
5.2.6.2	zmod4xxx_calc_rmox()	15
5.2.6.3	zmod4xxx_calc_single_rmox()	16
5.2.6.4	zmod4xxx_check_error_event()	16
5.2.6.5	zmod4xxx_init_measurement()	17
5.2.6.6	zmod4xxx_init_sensor()	17
5.2.6.7	zmod4xxx_null_ptr_check()	18
5.2.6.8	zmod4xxx_prepare_sensor()	18
5.2.6.9	zmod4xxx_read_adc_result()	19
5.2.6.10	zmod4xxx_read_rmox()	19
5.2.6.11	zmod4xxx_read_sensor_info()	20
5.2.6.12	zmod4xxx_read_status()	20
5.2.6.13	zmod4xxx_read_tracking_number()	21
5.2.6.14	zmod4xxx_start_measurement()	22
5.2.6.15	zmod4xxx_start_measurement_at()	22
5.3	HiCom Board API	23
5.3.1	Detailed Description	24
5.3.2	Data Structure Documentation	24
5.3.2.1	struct HiComInterface_t	24
5.3.3	Function Documentation	24

5.3.3.1	HiCom_Connect()	24
5.3.3.2	HiCom_Disconnect()	24
5.3.3.3	HiCom_Find()	25
5.3.3.4	HiCom_GetErrorString()	25
5.3.3.5	HiCom_I2CRead()	26
5.3.3.6	HiCom_I2CWrite()	26
5.3.3.7	HiCom_SetPower()	26
5.4	HSxxxx Sensor API	27
5.4.1	Detailed Description	27
5.4.2	Data Structure Documentation	28
5.4.2.1	struct HSxxxx_Results_t	28
5.4.2.2	struct HSxxxx_t	28
5.4.3	Function Documentation	29
5.4.3.1	HSxxxx_Init()	29
5.4.3.2	HSxxxx_Measure()	30
5.4.3.3	HSxxxx_Name()	30
5.5	HS4xxx Sensor API	32
5.5.1	Detailed Description	32
5.5.2	Enumeration Type Documentation	32
5.5.2.1	HS4xxx_ErrorCodes_t	32
5.5.3	Function Documentation	33
5.5.3.1	HS4xxx_Init()	33
5.5.3.2	HS4xxx_Measure()	33
5.5.3.3	HS4xxx_MeasureHold()	34
5.5.3.4	HS4xxx_MeasureRead()	35
5.5.3.5	HS4xxx_MeasureStart()	35
5.5.3.6	HS4xxx_ReadID()	36
5.6	HS3xxx Sensor API	37
5.6.1	Detailed Description	37

5.6.2	Enumeration Type Documentation	37
5.6.2.1	HS3xxx_ErrorCodes_t	37
5.6.3	Function Documentation	38
5.6.3.1	HS3xxx_Init()	38
5.6.3.2	HS3xxx_Measure()	38
5.6.3.3	HS3xxx_MeasureRead()	39
5.6.3.4	HS3xxx_MeasureStart()	40
5.6.3.5	HS3xxx_ReadID()	40
5.7	Hardware Abstraction Layer API	41
5.7.1	Detailed Description	42
5.7.2	Data Structure Documentation	42
5.7.2.1	struct Interface_t	42
5.7.3	Typedef Documentation	43
5.7.3.1	ErrorStringGenerator_t	43
5.7.3.2	I2CImpI_t	44
5.7.4	Enumeration Type Documentation	44
5.7.4.1	ErrorScope_t	44
5.7.4.2	GenericError_t	44
5.7.4.3	HALError_t	44
5.7.5	Function Documentation	45
5.7.5.1	HAL_Deinit()	45
5.7.5.2	HAL_GetErrorInfo()	45
5.7.5.3	HAL_GetErrorString()	46
5.7.5.4	HAL_HandleError()	47
5.7.5.5	HAL_Init()	47
5.7.5.6	HAL_SetError()	47
5.7.5.7	zmod4xxx_init()	48
5.8	ComBoard HAL	49
5.8.1	Detailed Description	49

5.9	Raspberry Pi HAL	50
5.9.1	Detailed Description	50
5.10	Custom HAL Template	51
5.11	ESCom Board API	52
5.11.1	Detailed Description	52
5.11.2	Data Structure Documentation	52
5.11.2.1	struct ESComInterface_t	52
5.11.3	Enumeration Type Documentation	53
5.11.3.1	ESComErrorScope_t	53
5.11.4	Function Documentation	53
5.11.4.1	ESCom_Connect()	53
5.11.4.2	ESCom_Disconnect()	53
5.11.4.3	ESCom_Find()	54
5.11.4.4	ESCom_GetErrorString()	54
5.11.4.5	ESCom_GetSensorVoltage()	55
5.11.4.6	ESCom_I2CRead()	55
5.11.4.7	ESCom_I2CWrite()	55
5.11.4.8	ESCom_SetPower()	55
5.12	Return codes of the algorithm functions.	57
5.12.1	Detailed Description	57
5.12.2	Macro Definition Documentation	57
5.12.2.1	ULP_O3_DAMAGE	57
5.12.2.2	ULP_O3_OK	57
5.12.2.3	ULP_O3_STABILIZATION	57

6	Data Structure Documentation	58
6.1	algorithm_version Struct Reference	58
6.1.1	Detailed Description	58
6.2	ulp_o3_handle_t Struct Reference	58
6.2.1	Detailed Description	59
6.2.2	Field Documentation	59
6.2.2.1	gcda	59
6.2.2.2	sample_cnt	59
6.3	ulp_o3_inputs_t Struct Reference	59
6.3.1	Detailed Description	59
6.4	ulp_o3_results_t Struct Reference	60
6.4.1	Detailed Description	60
6.4.2	Field Documentation	60
6.4.2.1	EPA_AQI	60
6.4.2.2	FAST_AQI	60
6.4.2.3	O3_conc_ppb	60
6.4.2.4	rmox	60
6.4.2.5	temperature	60
7	File Documentation	61
7.1	algos/ulp_o3.h File Reference	61
7.1.1	Function Documentation	62
7.1.1.1	calc_ulp_o3()	62
7.1.1.2	init_ulp_o3()	62
7.2	algos/zmod4510_config_ulp_o3.h File Reference	62
7.2.1	Variable Documentation	63
7.2.1.1	data_set_4510_init	63
7.2.1.2	data_set_4510_ulp_o3	63
7.2.1.3	zmod_ulp_o3_sensor_cfg	64
7.3	algos/zmod4xxx_cleaning.h File Reference	64
7.3.1	Function Documentation	64
7.3.1.1	zmod4xxx_cleaning_run()	64
7.4	hal/comboard/escom.h File Reference	65
7.5	hal/comboard/hicom.h File Reference	65
7.6	hal/hal.h File Reference	66
7.7	hal/raspi/rpi.h File Reference	67
7.8	hal/zmod4xxx_hal.h File Reference	67
7.9	sensors/hs3xxx.h File Reference	68
7.10	sensors/hs4xxx.h File Reference	68
7.11	sensors/hsxxxx.h File Reference	69
7.12	sensors/zmod4xxx.h File Reference	69
7.13	sensors/zmod4xxx_types.h File Reference	70

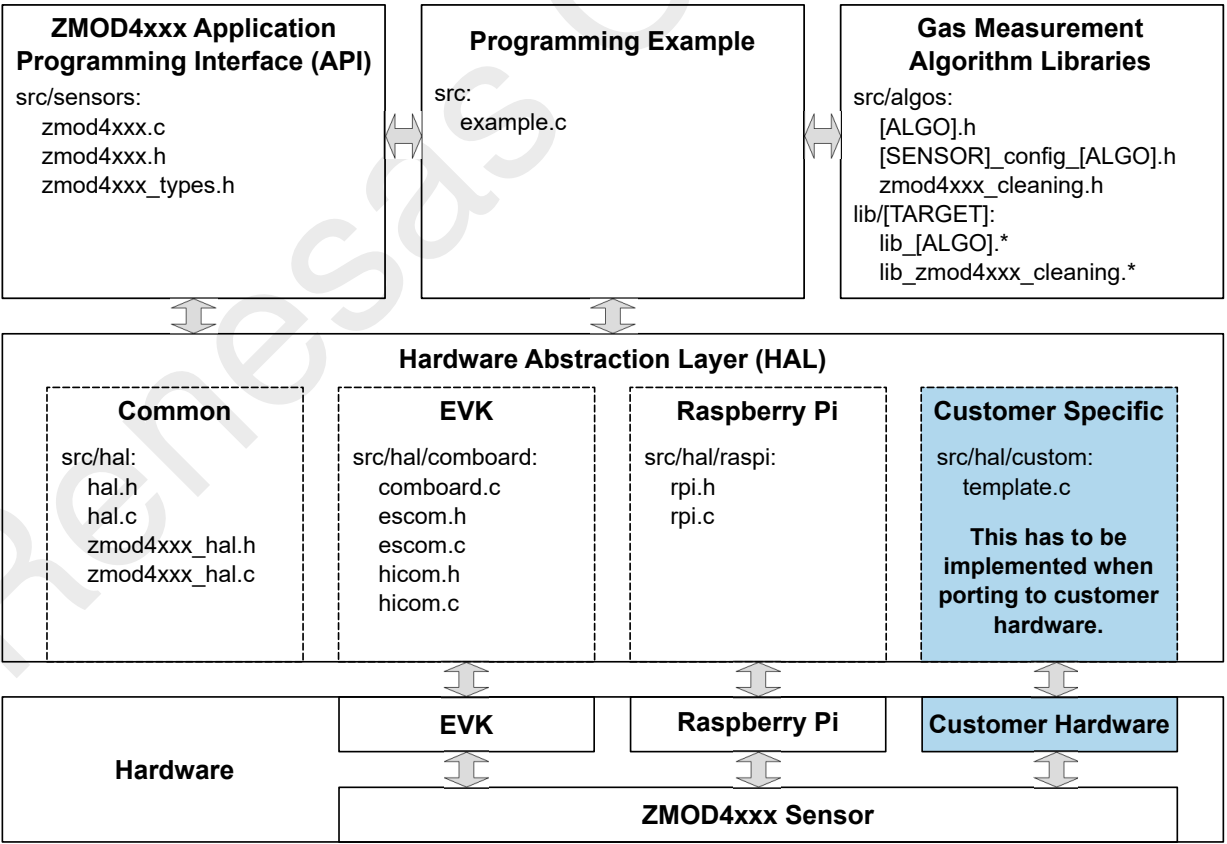
Chapter 1

Overview

This document describes the firmware for the ZMOD4510 gas sensor module using the ultra-low power mode (ULP) to measure ozone (O3).

The firmware package includes an example that can be compiled for the use with an EVK board which is connected to a host computer or for an Raspberry Pi. In addition an Arduino library is provided, that allows using the gas sensor on the Arduino MKRZERO.

The figure below shows an overview of the ZMOD4xxx API, programming example and libraries.



If required, the example can be adapted to run on customer specific hardware. For the adaption to the customer hardware, this firmware packages provides a HAL template in `hal/custom/template.c`. Please refer to the documentation of this file, and the [HAL API](#) documentation. Note that some of the HAL functions may not be required by specific sensors. Please refer to the *HAL API Requirements* section of the sensor you're trying to interface. Refer to the ZMOD4510 Programming Manual - Read Me for further information regarding sample code.

1.1 Folder Structure

- `src`: Source code directory, including a Makefile for below mentioned binaries
 - `sensors`: Headers and sources for different sensors
 - `hal`: Headers, sources and libraries for hardware abstraction
 - `algorithms`: Headers for algorithm and cleaning library
- `lib`: Precompiled algorithm libraries
- `doc`: Documentation
- `windows`: Pre-compiled example application for the EVK board on a 64-bit Windows platform
- `raspberrypi`: Pre-compiled example application for Raspberry-Pi, using the PiGPIO library
- `arduino`: Arduino library for installation in Arduino IDE (including fully working example code)

Note

Root privileges are required to execute the Raspberry-Pi example program.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Ultra-Low Power Ozone Library API 6

ZMOD4xxx Sensor API 7

HSxxxx Sensor API 27

 HS4xxx Sensor API 32

 HS3xxx Sensor API 37

Hardware Abstraction Layer API 41

 ComBoard HAL 49

 HiCom Board API 23

 ESCom Board API 52

 Raspberry Pi HAL 50

 Custom HAL Template 51

Return codes of the algorithm functions. 57

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

algorithm_version	Variables that describe the library version	58
ulp_o3_handle_t	Variables that describe the sensor or the algorithm state	58
ulp_o3_inputs_t	Variables that are needed for algorithm	59
ulp_o3_results_t	Variables that receive the algorithm outputs	60

Chapter 4

File Index

4.1 File List

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

algos/ulp_o3.h	This file contains the data structure definitions and the function definitions for the ULP O3 algorithm	61
algos/zmod4510_config_ulp_o3.h	This is the configuration for ZMOD4510 module - ulp_o3 library	62
algos/zmod4xxx_cleaning.h	This file contains the cleaning function definition for ZMOD4xxx	64
hal/hal.h	Renesas Environmental Sensor HAL definitions	66
hal/zmod4xxx_hal.h	ZMOD4xxx specific hardware abstraction layer definitions	67
hal/comboard/escom.h	ESCom board type and function declarations	65
hal/comboard/hicom.h	HiCom board type and function declarations	65
hal/raspi/rpi.h	Raspberry PI HAL type and function declarations	67
sensors/hs3xxx.h	HS3xxx sensor declarations	68
sensors/hs4xxx.h	HS4xxx sensor declarations	68
sensors/hsxxxx.h	Renesas humidity sensors (HS3xxx, HS4xxx) abstraction	69
sensors/zmod4xxx.h	Zmod4xxx-API functions	69
sensors/zmod4xxx_types.h	Zmod4xxx types	70

Chapter 5

Module Documentation

5.1 Ultra-Low Power Ozone Library API

5.2 ZMOD4xxx Sensor API

Files

- file [zmod4xxx.h](#)
zmod4xxx-API functions
- file [zmod4xxx_types.h](#)
zmod4xxx types

Functions

- [zmod4xxx_err zmod4xxx_calc_factor](#) ([zmod4xxx_conf](#) *conf, uint8_t *hsp, uint8_t *config)
Calculate measurement settings.
- [float zmod4xxx_calc_single_rmx](#) ([zmod4xxx_dev_t](#) *dev, uint8_t *adc_result)
Calculate rmx resistance from ADC raw data.
- [zmod4xxx_err zmod4xxx_calc_rmx](#) ([zmod4xxx_dev_t](#) *dev, uint8_t *adc_result, float *rmx)
Calculate rmx resistance on array of results.
- [zmod4xxx_err zmod4xxx_check_error_event](#) ([zmod4xxx_dev_t](#) *dev)
Check the error event of the device.
- [zmod4xxx_err zmod4xxx_init_measurement](#) ([zmod4xxx_dev_t](#) *dev)
Initialize the sensor for corresponding measurement.
- [zmod4xxx_err zmod4xxx_init_sensor](#) ([zmod4xxx_dev_t](#) *dev)
Initialize the sensor after power on.
- [zmod4xxx_err zmod4xxx_null_ptr_check](#) ([zmod4xxx_dev_t](#) *dev)
Check if all function pointers are assigned.
- [zmod4xxx_err zmod4xxx_prepare_sensor](#) ([zmod4xxx_dev_t](#) *dev)
High-level function to prepare sensor.
- [zmod4xxx_err zmod4xxx_read_adc_result](#) ([zmod4xxx_dev_t](#) *dev, uint8_t *adc_result)
Read adc values from the sensor.
- [zmod4xxx_err zmod4xxx_read_rmx](#) ([zmod4xxx_dev_t](#) *dev, uint8_t *adc_result, float *rmx)
High-level function to read rmx.
- [zmod4xxx_err zmod4xxx_read_sensor_info](#) ([zmod4xxx_dev_t](#) *dev)
Read sensor parameter.
- [zmod4xxx_err zmod4xxx_read_status](#) ([zmod4xxx_dev_t](#) *dev, uint8_t *status)
Read the status of the device.
- [zmod4xxx_err zmod4xxx_read_tracking_number](#) ([zmod4xxx_dev_t](#) *dev, uint8_t *track_num)
Read tracking number of sensor.
- [zmod4xxx_err zmod4xxx_start_measurement](#) ([zmod4xxx_dev_t](#) *dev)
Start the measurement.
- [zmod4xxx_err zmod4xxx_start_measurement_at](#) ([zmod4xxx_dev_t](#) *dev, uint8_t step)
Start the measurement at an user-defined sequencer step.

Data Structures

- struct [zmod4xxx_conf_str](#)
A single data set for the configuration. [More...](#)
- struct [zmod4xxx_conf](#)
Structure to hold the gas sensor module configuration. [More...](#)
- struct [zmod4xxx_dev_t](#)
Device structure ZMOD4xxx. [More...](#)

Macros

- `#define ZMOD4XXX_ADDR_PID (0x00)`
- `#define ZMOD4XXX_ADDR_CONF (0x20)`
- `#define ZMOD4XXX_ADDR_PROD_DATA (0x26)`
- `#define ZMOD4XXX_ADDR_CMD (0x93)`
- `#define ZMOD4XXX_ADDR_STATUS (0x94)`
- `#define ZMOD4XXX_ADDR_TRACKING (0x3A)`
- `#define ZMOD4XXX_LEN_PID (2)`
- `#define ZMOD4XXX_LEN_CONF (6)`
- `#define ZMOD4XXX_LEN_TRACKING (6)`
- `#define HSP_MAX (8)`
- `#define RSLT_MAX (32)`
- `#define STATUS_SEQUENCER_RUNNING_MASK (0x80)`
- `#define STATUS_SLEEP_TIMER_ENABLED_MASK (0x40)`
- `#define STATUS_ALARM_MASK (0x20)`
- `#define STATUS_LAST_SEQ_STEP_MASK (0x1F)`
- `#define STATUS_POR_EVENT_MASK (0x80)`
- `#define STATUS_ACCESS_CONFLICT_MASK (0x40)`

Typedefs

- `typedef int8_t(* zmod4xxx_i2c_ptr_t) (uint8_t addr, uint8_t reg_addr, uint8_t *data_buf, uint8_t len)`
function pointer type for i2c access
- `typedef void(* zmod4xxx_delay_ptr_p) (uint32_t ms)`
function pointer to hardware dependent delay function

Enumerations

- `enum zmod4xxx_err {`
`ZMOD4XXX_OK = 0, ERROR_INIT_OUT_OF_RANGE, ERROR_GAS_TIMEOUT, ERROR_I2C = -3,`
`ERROR_SENSOR_UNSUPPORTED, ERROR_CONFIG_MISSING, ERROR_ACCESS_CONFLICT, ERR←`
`OR_POR_EVENT,`
`ERROR_CLEANING, ERROR_NULL_PTR }`
error_codes Error codes

5.2.1 Detailed Description

All ZMOD4xxx based gas sensing applications generate their results using algorithm libraries which are computing the desired result from raw data input that is delivered by the gas sensor. An overview of different algorithm implementations is given in *ZMOD4xxx Programming Manual - Read Me*. The raw sensor data is obtained through the ZMOD4xxx API. This API defines data structures and functions required to configure and operate the sensor. All of these functions work by accessing the sensor through its I2C interface.

As the sensor may be used in arbitrary hardware environments, the ZMOD4xxx API requires a hardware abstraction layer (HAL), providing access to hardware specific functions in a generic way. The HAL minimizes the effort to port a ZMOD4xxx application to a new platform (e.g. MCU). Only the HAL related files need to be provided.

An overview on the relation of the different sensor system components is given in the figure on the Overview section.

HAL API ports to a customer specific hardware require the following function pointers of the [Interface_t](#) HAL data structure to be initialized and working as documented:

Function Pointer	Required
i2cRead	Mandatory
i2cWrite	Mandatory
msSleep	Mandatory
reset	Not required

5.2.2 Data Structure Documentation

5.2.2.1 struct zmod4xxx_conf_str

A single data set for the configuration.

Data Fields

- uint8_t **addr**
- uint8_t **len**
- uint8_t * **data_buf**

5.2.2.2 struct zmod4xxx_conf

Structure to hold the gas sensor module configuration.

Data Fields

- uint8_t **start**
- [zmod4xxx_conf_str](#) **h**
- [zmod4xxx_conf_str](#) **d**
- [zmod4xxx_conf_str](#) **m**
- [zmod4xxx_conf_str](#) **s**
- [zmod4xxx_conf_str](#) **r**
- uint8_t **prod_data_len**

5.2.2.3 struct zmod4xxx_dev_t

Device structure ZMOD4xxx.

Data Fields

- uint8_t [i2c_addr](#)
- uint8_t [config](#) [6]
- uint16_t [mox_er](#)
- uint16_t [mox_lr](#)
- uint16_t [pid](#)
- uint8_t * [prod_data](#)
- [zmod4xxx_i2c_ptr_t](#) [read](#)
- [zmod4xxx_i2c_ptr_t](#) [write](#)
- [zmod4xxx_delay_ptr_p](#) [delay_ms](#)
- [zmod4xxx_conf](#) * [init_conf](#)
- [zmod4xxx_conf](#) * [meas_conf](#)

5.2.2.3.1 Field Documentation

5.2.2.3.1.1 config

```
uint8_t config[6]
```

configuration parameter set

5.2.2.3.1.2 delay_ms

```
zmod4xxx_delay_ptr_p delay_ms
```

function pointer to delay function

5.2.2.3.1.3 i2c_addr

```
uint8_t i2c_addr
```

i2c address of the sensor

5.2.2.3.1.4 init_conf

```
zmod4xxx_conf* init_conf
```

pointer to the init configuration

5.2.2.3.1.5 meas_conf

```
zmod4xxx_conf* meas_conf
```

pointer to the measurement configuration

5.2.2.3.1.6 mox_er

```
uint16_t mox_er
```

sensor specific parameter

5.2.2.3.1.7 mox_lr

```
uint16_t mox_lr
```

sensor specific parameter

5.2.2.3.1.8 pid

```
uint16_t pid
```

product id of the sensor

5.2.2.3.1.9 prod_data

uint8_t* prod_data

production data

5.2.2.3.1.10 read

zmod4xxx_i2c_ptr_t read

function pointer to i2c read

5.2.2.3.1.11 write

zmod4xxx_i2c_ptr_t write

function pointer to i2c write

5.2.3 Macro Definition Documentation

5.2.3.1 STATUS_ACCESS_CONFLICT_MASK

```
#define STATUS_ACCESS_CONFLICT_MASK (0x40)
```

AccessConflict

5.2.3.2 STATUS_ALARM_MASK

```
#define STATUS_ALARM_MASK (0x20)
```

Alarm

5.2.3.3 STATUS_LAST_SEQ_STEP_MASK

```
#define STATUS_LAST_SEQ_STEP_MASK (0x1F)
```

Last executed sequencer step

5.2.3.4 STATUS_POR_EVENT_MASK

```
#define STATUS_POR_EVENT_MASK (0x80)
```

POR_event

5.2.3.5 STATUS_SEQUENCER_RUNNING_MASK

```
#define STATUS_SEQUENCER_RUNNING_MASK (0x80)
```

Sequencer is running

5.2.3.6 STATUS_SLEEP_TIMER_ENABLED_MASK

```
#define STATUS_SLEEP_TIMER_ENABLED_MASK (0x40)
```

SleepTimer_enabled

5.2.4 Typedef Documentation

5.2.4.1 zmod4xxx_delay_ptr_p

```
typedef void(* zmod4xxx_delay_ptr_p) (uint32_t ms)
```

function pointer to hardware dependent delay function

Parameters

in	<i>delay</i>	in milliseconds
----	--------------	-----------------

Returns

none

5.2.4.2 zmod4xxx_i2c_ptr_t

```
typedef int8_t(* zmod4xxx_i2c_ptr_t) (uint8_t addr, uint8_t reg_addr, uint8_t *data_buf, uint8_t len)
```

function pointer type for i2c access

Parameters

in	<i>addr</i>	7-bit I2C slave address of the ZMOD4xxx
in	<i>reg_addr</i>	address of internal register to read/write
in, out	<i>data</i>	pointer to the read/write data value
in	<i>len</i>	number of bytes to read/write

Returns

error code

Return values

0	success
!= 0	error

5.2.5 Enumeration Type Documentation

5.2.5.1 zmod4xxx_err

enum zmod4xxx_err

error_codes Error codes

Enumerator

ERROR_INIT_OUT_OF_RANGE	The initialization value is out of range.
ERROR_GAS_TIMEOUT	A previous measurement is running that could not be stopped or sensor does not respond.
ERROR_I2C	I2C communication was not successful.
ERROR_SENSOR_UNSUPPORTED	The Firmware configuration used does not match the sensor module.
ERROR_CONFIG_MISSING	There is no pointer to a valid configuration.
ERROR_ACCESS_CONFLICT	Invalid ADC results due to a still running measurement while results readout.
ERROR_POR_EVENT	Power-on reset event. Check power supply and reset pin.
ERROR_CLEANING	The maximum numbers of cleaning cycles ran on this sensor. Cleaning function has no effect anymore.
ERROR_NULL_PTR	The dev structure did not receive the pointers for I2C read, write and/or delay.

5.2.6 Function Documentation

5.2.6.1 zmod4xxx_calc_factor()

```
zmod4xxx_err zmod4xxx_calc_factor (  
    zmod4xxx_conf * conf,
```

```
uint8_t * hsp,  
uint8_t * config )
```

Calculate measurement settings.

Renesas Confidential

Parameters

in	<i>conf</i>	measurement configuration data
in	<i>hsp</i>	heater set point pointer
in	<i>config</i>	sensor configuration data pointer

Returns

error code

Return values

0	success
---	---------

5.2.6.2 zmod4xxx_calc_rmx()

```
zmod4xxx_err zmod4xxx_calc_rmx (
    zmod4xxx_dev_t * dev,
    uint8_t * adc_result,
    float * rmx )
```

Calculate rmx resistance on array of results.

Note

This is not a generic function. Only use it if indicated in your example program flow.
This function uses zmod4xxx_calc_single_rmx

Parameters

in	<i>dev</i>	pointer to the device
in, out	<i>adc_result</i>	pointer to the adc results
in, out	<i>rmx</i>	pointer to the rmx values

Returns

error code

Return values

0	success
!= 0	error

5.2.6.3 zmod4xxx_calc_single_rmox()

```
float zmod4xxx_calc_single_rmox (
    zmod4xxx_dev_t * dev,
    uint8_t * adc_result )
```

Calculate mox resistance from ADC raw data.

Note
This is not a generic function. Only use it if indicated in your example program flow.

Parameters

in	dev	pointer to the device
in, out	adc_result	pointer to the adc results

Returns

computed MOX resistance

5.2.6.4 zmod4xxx_check_error_event()

```
zmod4xxx_err zmod4xxx_check_error_event (
    zmod4xxx_dev_t * dev )
```

Check the error event of the device.

Parameters

in	dev	pointer to the device
----	-----	-----------------------

Returns

error code

Return values

0	success
!= 0	error

5.2.6.5 zmod4xxx_init_measurement()

```
zmod4xxx_err zmod4xxx_init_measurement (
    zmod4xxx_dev_t * dev )
```

Initialize the sensor for corresponding measurement.

Parameters

in	dev	pointer to the device
----	-----	-----------------------

Returns

error code

Return values

0	success
!= 0	error

Note

Before calling function, measurement data set has to be passed the dev->meas_conf

5.2.6.6 zmod4xxx_init_sensor()

```
zmod4xxx_err zmod4xxx_init_sensor (
    zmod4xxx_dev_t * dev )
```

Initialize the sensor after power on.

Parameters

in	dev	pointer to the device
----	-----	-----------------------

Returns

error code

Return values

0	success
!= 0	error

Note

Before calling function, initialization data set has to be passed the dev->init_conf

5.2.6.7 zmod4xxx_null_ptr_check()

```
zmod4xxx_err zmod4xxx_null_ptr_check (
    zmod4xxx_dev_t * dev )
```

Check if all function pointers are assigned.

Parameters

in	dev	pointer to the device
----	-----	-----------------------

Returns

error code

Return values

0	success
!= 0	error

5.2.6.8 zmod4xxx_prepare_sensor()

```
zmod4xxx_err zmod4xxx_prepare_sensor (
    zmod4xxx_dev_t * dev )
```

High-level function to prepare sensor.

Parameters

in	dev	pointer to the device
----	-----	-----------------------

Returns

error code

Return values

0	success
!=0	error

5.2.6.9 zmod4xxx_read_adc_result()

```
zmod4xxx_err zmod4xxx_read_adc_result (
    zmod4xxx_dev_t * dev,
    uint8_t * adc_result )
```

Read adc values from the sensor.

Parameters

in	dev	pointer to the device
in, out	adc_result	pointer to the adc results

Returns

error code

Return values

0	success
!= 0	error

5.2.6.10 zmod4xxx_read_rmx()

```
zmod4xxx_err zmod4xxx_read_rmx (
    zmod4xxx_dev_t * dev,
    uint8_t * adc_result,
    float * rmx )
```

High-level function to read rmx.

Note

This is not a generic function. Only use it if indicated in your example program flow.

Parameters

in	dev	pointer to the device
in, out	adc_result	pointer to the adc results
in, out	rmx	pointer to the rmx values

Returns

error code

Return values

0	success
!= 0	error

5.2.6.11 zmod4xxx_read_sensor_info()

```
zmod4xxx_err zmod4xxx_read_sensor_info (  
    zmod4xxx_dev_t * dev )
```

Read sensor parameter.

Parameters

in	dev	pointer to the device
----	-----	-----------------------

Returns

error code

Return values

0	success
!= 0	error

Note

This function must be called once before running other sensor functions.

5.2.6.12 zmod4xxx_read_status()

```
zmod4xxx_err zmod4xxx_read_status (  
    zmod4xxx_dev_t * dev,  
    uint8_t * status )
```

Read the status of the device.

Parameters

<i>in</i>	<i>dev</i>	pointer to the device
<i>in, out</i>	<i>status</i>	pointer to the status variable

Returns

error code

Return values

<i>0</i>	success
<i>!= 0</i>	error

5.2.6.13 zmod4xxx_read_tracking_number()

```
zmod4xxx_err zmod4xxx_read_tracking_number (
    zmod4xxx_dev_t * dev,
    uint8_t * track_num )
```

Read tracking number of sensor.

Note

The buffer pointed to by track_num must be at least 6 bytes long

Parameters

<i>in</i>	<i>dev</i>	pointer to the device
<i>in, out</i>	<i>track_num</i>	pointer to buffer to store the tracking number

Returns

error code

Return values

<i>0</i>	success
<i>!= 0</i>	error

5.2.6.14 `zmod4xxx_start_measurement()`

```
zmod4xxx_err zmod4xxx_start_measurement (
    zmod4xxx_dev_t * dev )
```

Start the measurement.

Parameters

in	<i>dev</i>	pointer to the device
----	------------	-----------------------

Returns

error code

Return values

0	success
<i>!= 0</i>	error

5.2.6.15 `zmod4xxx_start_measurement_at()`

```
zmod4xxx_err zmod4xxx_start_measurement_at (
    zmod4xxx_dev_t * dev,
    uint8_t step )
```

Start the measurement at an user-defined sequencer step.

Parameters

in	<i>dev</i>	pointer to the device
in	<i>step</i>	sequencer step to start at

Returns

error code

Return values

0	success
<i>!= 0</i>	error

5.3 HiCom Board API

Files

- file [hicom.h](#)
HiCom board type and function declarations.

Functions

- int [HiCom_Find](#) ([HiComInterface_t](#) *board, int *count)
Enumerate HiCom boards Scans USB ports for connected HiCom boards.
- int [HiCom_Connect](#) ([HiComInterface_t](#) *board)
Connect a HiCom instance Tries to connect to an interface that has been discovered with [HiCom_Find\(\)](#) previously.
- int [HiCom_Disconnect](#) ([HiComInterface_t](#) *board)
Disconnect a HicomBoard.
- int [HiCom_SetPower](#) ([HiComInterface_t](#) *board, bool on)
Switch sensor power supply on or off.
- int [HiCom_I2CWrite](#) ([HiComInterface_t](#) *board, uint8_t slAddr, uint8_t *wData1, int wSize1, uint8_t *wData2, int wSize2)
- int [HiCom_I2CRead](#) ([HiComInterface_t](#) *board, uint8_t slAddr, uint8_t *wData, int wSize, uint8_t *rData, int rSize)
- char const * [HiCom_GetErrorString](#) (int error, int scope, char *buf, int bufLen)

Data Structures

- struct [HiComInterface_t](#)

Macros

- **#define HICOM_NAME** "Dual RS232-HS A"
- **#define HICOM_I2C_SPEED** 100000

Typedefs

- typedef FT_STATUS **HiComStatus_t**
- typedef FT_HANDLE **HiComHandle_t**

Enumerations

- enum **HiComErrorScope_t** { **iesFTDI** = 0x21 }

5.3.1 Detailed Description

5.3.2 Data Structure Documentation

5.3.2.1 struct HiComInterface_t

Data Fields

- FT_DEVICE_LIST_INFO_NODE **node**
- int **index**

5.3.3 Function Documentation

5.3.3.1 HiCom_Connect()

```
int HiCom_Connect (
    HiComInterface_t * board )
```

Connect a HiCom instance Tries to connect to an interface that has been discovered with [HiCom_Find\(\)](#) previously.

Parameters

<i>board</i>	Pointer to HiCom board discovered by HiCom_Find()
--------------	---

Returns

int 0 on success, error code otherwise

5.3.3.2 HiCom_Disconnect()

```
int HiCom_Disconnect (
    HiComInterface_t * board )
```

Disconnect a HicomBoard.

Parameters

<i>board</i>	Pointer to HiCom board to be disconnected
--------------	---

Returns

int

5.3.3.3 HiCom_Find()

```
int HiCom_Find (
    HiComInterface_t * board,
    int * count )
```

Enumerate HiCom boards Scans USB ports for connected HiCom boards.

Parameters

<i>board</i>	pointer to a buffer storing board information
<i>count</i>	[in] maximum count of boards to be stored [out] actual number of boards stored

Returns

int 0 on success, error code otherwise

5.3.3.4 HiCom_GetErrorString()

```
char const* HiCom_GetErrorString (
    int error,
    int scope,
    char * buf,
    int bufLen )
```

Generate a descriptive error message

Parameters

in	<i>error</i>	Error code
in	<i>scope</i>	Error scope

Returns

Error string

5.3.3.5 HiCom_I2CRead()

```
int HiCom_I2CRead (
    HiComInterface_t * board,
    uint8_t sIAddr,
    uint8_t * wData,
    int wSize,
    uint8_t * rData,
    int rSize )
```

HiCom implementation of HAL_t::i2cRead

5.3.3.6 HiCom_I2CWrite()

```
int HiCom_I2CWrite (
    HiComInterface_t * board,
    uint8_t sIAddr,
    uint8_t * wData1,
    int wSize1,
    uint8_t * wData2,
    int wSize2 )
```

HiCom implementation of HAL_t::i2cWrite

5.3.3.7 HiCom_SetPower()

```
int HiCom_SetPower (
    HiComInterface_t * board,
    bool on )
```

Switch sensor power supply on or off.

Parameters

<i>board</i>	Pointer to HiCom board to be operated
--------------	---------------------------------------

Returns

int 0 on success, error code otherwise

5.4 HSxxx Sensor API

API providing a unified interface for Renesas Humidity and Temperature sensors.

Modules

- [HS4xxx Sensor API](#)
HS4xxx Temperature/Humidity Sensor API.
- [HS3xxx Sensor API](#)
HS3xxx Temperature/Humidity Sensor API.

Files

- file [hsxxxx.h](#)
Renesas humidity sensors (HS3xxx, HS4xxx) abstraction.

Functions

- `int HSxxxx_Init (HSxxxx_t *sensor, Interface_t *hal)`
Initialize the sensor object.
- `int HSxxxx_Measure (HSxxxx_t *sensor, HSxxxx_Results_t *results)`
Perform one temperature measurement.
- `char const * HSxxxx_Name (HSxxxx_t *sensor)`
Return the temperature/humidity sensor type name.

Data Structures

- struct [HSxxxx_Results_t](#)
Data structure holding humidity/temperature results. [More...](#)
- struct [HSxxxx_t](#)
Data structure holding information required for HSxxxx API operation. [More...](#)

5.4.1 Detailed Description

API providing a unified interface for Renesas Humidity and Temperature sensors.

The Renesas HS3xxx and HS4xxx are highly accurate, ultra-low power, fully calibrated relative humidity and temperature sensors. These sensors may be used as standalone sensors or used as companion sensor for humidity and temperature sensitive applications (e.g., for Renesas gas sensors).

The present API is provided as a unified interface to both of these sensor types.

HAL API ports to a customer specific hardware require the following function pointers of the [Interface_t](#) HAL data structure to be initialized and working as documented:

Function Pointer	Required
i2cRead	Mandatory
i2cWrite	Mandatory
msSleep	Mandatory
reset	Not required

5.4.2 Data Structure Documentation

5.4.2.1 struct HSxxxx_Results_t

Data structure holding humidity/temperature results.

Data Fields

- float [temperature](#)
- float [humidity](#)

5.4.2.1.1 Field Documentation

5.4.2.1.1.1 humidity

float humidity

Relative humidity

5.4.2.1.1.2 temperature

float temperature

Temperature value in degree Celsius

5.4.2.2 struct HSxxxx_t

Data structure holding information required for HSxxxx API operation.

Data Fields

- [Interface_t](#) * [interface](#)
- uint8_t [i2cAddress](#)

5.4.2.2.1 Field Documentation

5.4.2.2.1.1 i2cAddress

uint8_t i2cAddress

I2C slave address of the HSxxxx sensor

5.4.2.2.1.2 interface

Interface_t* interface

Pointer to the hal object for physical communication

5.4.3 Function Documentation

5.4.3.1 HSxxxx_Init()

```
int HSxxxx_Init (
    HSxxxx_t * sensor,
    Interface_t * hal )
```

Initialize the sensor object.

This function tries searching for a HS4xxx sensor first, by accessing the corresponding I2C address. If no such sensor is found, the function searches for a HS3xxx.

The type of sensor being detected can be determined using the function [HSxxxx_Name\(\)](#).

Parameters

<i>sensor</i>	Pointer to sensor object to be initialized.
<i>hal</i>	Pointer to hal object for physical communication.

Returns

int Error code

Return values

<i>0</i>	On success
<i>other</i>	On error

5.4.3.2 HSxxx_Measure()

```
int HSxxx_Measure (
    HSxxx_t * sensor,
    HSxxx_Results_t * results )
```

Perform one temperature measurement.

This function starts a temperature/humidity measurement and waits for the availability of the result before it returns.

Note

This function is implemented as blocking function. Thus while the measurement is ongoing no other code is executed. Depending on the sensor the bocking time may be multiple tens of milliseconds.

Parameters

<i>sensor</i>	Pointer to the sensor object to be used.
<i>results</i>	Pointer to data structure for result storage.

Returns

int Error code

Return values

<i>0</i>	On success
<i>other</i>	On error

5.4.3.3 HSxxx_Name()

```
char const* HSxxx_Name (
    HSxxx_t * sensor )
```

Return the temperature/humidity sensor type name.

The function HSxxx_Init can identify different types of temperature/ humidity sensors. This function may be used to determine the sensor type that has been identified. The identification is performed based on the sensors I2C address.

Parameters

<i>sensor</i>	Pointer to the sensor object to be queried.
---------------	---

Returns

int Error code

Return values

0	On success
other	On error

5.5 HS4xxx Sensor API

HS4xxx Temperature/Humidity Sensor API.

Files

- file [hs4xxx.h](#)
HS4xxx sensor declarations.

Functions

- int [HS4xxx_Init](#) ([HSxxxx_t](#) *sensor, [Interface_t](#) *hal)
Initialize the sensor object.
- int [HS4xxx_ReadID](#) ([HSxxxx_t](#) *sensor, uint32_t *id)
Read the unique sensor ID of the HS4xxx.
- int [HS4xxx_Measure](#) ([HSxxxx_t](#) *sensor, [HSxxxx_Results_t](#) *results)
Perform a temperature/humidity measurement cycle.
- int [HS4xxx_MeasureHold](#) ([HSxxxx_t](#) *sensor, [HSxxxx_Results_t](#) *results)
Perform a temperature/humidity measurement cycle using hold mode.
- int [HS4xxx_MeasureStart](#) ([HSxxxx_t](#) *sensor)
Start a temperature/humidity measurement cycle in non-hold mode.
- int [HS4xxx_MeasureRead](#) ([HSxxxx_t](#) *sensor, [HSxxxx_Results_t](#) *results)
Read temperature/humidity results in non-hold mode.

Enumerations

- enum [HS4xxx_ErrorCodes_t](#) { [hteHS4xxxCRCError](#) = 1 }
Error code definitions specific for the HS4xxx API.

5.5.1 Detailed Description

HS4xxx Temperature/Humidity Sensor API.

HS4xxx is a series of highly accurate, fully calibrated automotive-grade relative humidity and temperature sensors. HS4xxx sensors provide a CRC checksum for communication integrity checking and have different modes of operation.

The HS4xxx API provides convenient access to the sensor's temperature & humidity measurement capabilities.

5.5.2 Enumeration Type Documentation

5.5.2.1 HS4xxx_ErrorCodes_t

enum [HS4xxx_ErrorCodes_t](#)

Error code definitions specific for the HS4xxx API.

Enumerator

hteHS4xxxCRCError	Result CRC error
-------------------	------------------

5.5.3 Function Documentation

5.5.3.1 HS4xxx_Init()

```
int HS4xxx_Init (
    HSxxxx_t * sensor,
    Interface_t * hal )
```

Initialize the sensor object.

This function tries accessing the HS4xxx I2C address using *hal* as hardware interface. On success, the sensor object is initialized and can be used afterwards. Otherwise, an error code is returned and the sensor object is not usable.

Note

The HS4xxx API requires that the HAL interface object has the [Interface_t::i2cRead](#) and [Interface_t::i2cWrite](#) members defined. In addition, [HS4xxx_Measure\(\)](#) requires the [Interface_t::msSleep](#) member.

Parameters

<i>sensor</i>	Pointer to sensor object to be initialized
<i>hal</i>	Pointer to HAL object providing physical communication

Returns

int Error code

Return values

0	On success
other	On error

5.5.3.2 HS4xxx_Measure()

```
int HS4xxx_Measure (
    HSxxxx_t * sensor,
    HSxxxx_Results_t * results )
```

Perform a temperature/humidity measurement cycle.

This function starts a measurement cycle in non-hold mode, waits for the result to be available and reads it. If the checksum computation is correct the result is stored in the *results* data structure. Otherwise the contents of *results* is left unmodified and an error code is returned.

Note

Although the HS4xxx is able to perform the requested measurement with a single I2C transaction (refer to H↔S4xxx_MeasureHold), this function uses the [HS4xxx_MeasureStart\(\)](#) and [HS4xxx_MeasureRead\(\)](#) functions together with the [Interface_t::msSleep](#) function. This is to allow operation on a wider variety of I2C interfaces. [HS4xxx_MeasureHold\(\)](#) requires a minimum of 200kHz I2C clock frequency which is not supported by all interfaces.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
<i>results</i>	Pointer to a data structure, to store results in

Returns

int Error code

Return values

0	On success
other	On error

5.5.3.3 HS4xxx_MeasureHold()

```
int HS4xxx_MeasureHold (
    HSxxxx_t * sensor,
    HSxxxx_Results_t * results )
```

Perform a temperature/humidity measurement cycle using hold mode.

This function reads temperature/humidity values from the HS4xxx in hold mode (Refer to the HS4xxx datasheet for an explanation of hold mode). If the checksum computation is correct the result is stored in the *results* data structure. Otherwise the contents of *results* is left unmodified and an error code is returned.

Note

For hold measurements, the I2C clock frequency must be at least 200kHz. Otherwise the result readout is unreliable.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
<i>results</i>	Pointer to a data structure, to store results in

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

5.5.3.4 HS4xxx_MeasureRead()

```
int HS4xxx_MeasureRead (
    HSxxxx_t * sensor,
    HSxxxx_Results_t * results )
```

Read temperature/humidity results in non-hold mode.

This function reads the temperature/humidity results from a measurement that has been started through [HS4xxx↵_MeasureStart\(\)](#) previously. If the checksum computation is correct the result is stored in the *results* data structure. Otherwise the contents of *results* is left unmodified and an error code is returned.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
<i>results</i>	Pointer to a data structure, to store results in

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

5.5.3.5 HS4xxx_MeasureStart()

```
int HS4xxx_MeasureStart (
```

```
HSxxxx_t * sensor )
```

Start a temperature/humidity measurement cycle in non-hold mode.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
---------------	---

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

5.5.3.6 HS4xxx_ReadID()

```
int HS4xxx_ReadID (
    HSxxxx_t * sensor,
    uint32_t * id )
```

Read the unique sensor ID of the HS4xxx.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
<i>id</i>	Pointer buffer to store the ID

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

5.6 HS3xxx Sensor API

HS3xxx Temperature/Humidity Sensor API.

Files

- file [hs3xxx.h](#)
HS3xxx sensor declarations.

Functions

- int [HS3xxx_Init](#) ([HSxxxx_t](#) *sensor, [Interface_t](#) *hal)
Initialize the sensor object.
- int [HS3xxx_ReadID](#) ([HSxxxx_t](#) *sensor, uint32_t *id)
Read the unique sensor ID of the HS3xxx.
- int [HS3xxx_Measure](#) ([HSxxxx_t](#) *sensor, [HSxxxx_Results_t](#) *results)
Execute a temperature/humidity measurement cycle.
- int [HS3xxx_MeasureStart](#) ([HSxxxx_t](#) *sensor)
Start a temperature/humidity measurement cycle.
- int [HS3xxx_MeasureRead](#) ([HSxxxx_t](#) *sensor, [HSxxxx_Results_t](#) *results)
Read temperature/humidity results.

Enumerations

- enum [HS3xxx_ErrorCodes_t](#) { [hteStaleData](#) = 1 }
Error code definitions specific for the HS3xxx API.

5.6.1 Detailed Description

HS3xxx Temperature/Humidity Sensor API.

HS3xxx is a series of highly-accurate, fully-calibrated relative humidity and temperature sensors.

The HS3xxx API provides convenient access to the temperature & humidity measurement capabilities of the sensor.

5.6.2 Enumeration Type Documentation

5.6.2.1 HS3xxx_ErrorCodes_t

enum [HS3xxx_ErrorCodes_t](#)

Error code definitions specific for the HS3xxx API.

Enumerator

<code>hteStaleData</code>	Sensor reported stale data.
---------------------------	-----------------------------

5.6.3 Function Documentation

5.6.3.1 HS3xxx_Init()

```
int HS3xxx_Init (
    HSxxxx_t * sensor,
    Interface_t * hal )
```

Initialize the sensor object.

This function tries accessing the HS3xxx I2C address using *hal* as hardware interface. On success, the sensor object is initialized and can be used afterwards. Otherwise, an error code is returned and the sensor object is not usable.

Note

The HS3xxx API requires that the HAL interface object has the [Interface_t::i2cRead](#) and [Interface_t::i2cWrite](#) members defined. In addition, [HS3xxx_Measure\(\)](#) requires the [Interface_t::msSleep](#) member, however, the check for this function is done in [HS3xxx_Measure\(\)](#) and not in [HS3xxx_Init\(\)](#).

Parameters

<i>sensor</i>	Pointer to sensor object to be initialized
<i>hal</i>	Pointer to HAL object providing physical communication

Returns

int Error code

Return values

<code>0</code>	On success
<i>other</i>	On error

5.6.3.2 HS3xxx_Measure()

```
int HS3xxx_Measure (
    HSxxxx_t * sensor,
    HSxxxx_Results_t * results )
```

Execute a temperature/humidity measurement cycle.

This function starts a measurement cycle in non-hold mode, waits for the result to be available and reads it. This is a convenience function which calls [HS3xxx_MeasureStart\(\)](#) and [HS3xxx_MeasureRead\(\)](#) with a delay between both calls, to allow the sensor to complete its measurement.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
<i>results</i>	Pointer to a data structure, to store results in

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

5.6.3.3 HS3xxx_MeasureRead()

```
int HS3xxx_MeasureRead (
    HSxxxx_t * sensor,
    HSxxxx_Results_t * results )
```

Read temperature/humidity results.

This function reads the temperature/humidity results from a measurement that has been started through [HS3xxx_MeasureStart\(\)](#) previously.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
<i>results</i>	Pointer to a data structure, to store results in

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

5.6.3.4 HS3xxx_MeasureStart()

```
int HS3xxx_MeasureStart (
    HSxxxx_t * sensor )
```

Start a temperature/humidity measurement cycle.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
---------------	---

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

5.6.3.5 HS3xxx_ReadID()

```
int HS3xxx_ReadID (
    HSxxxx_t * sensor,
    uint32_t * id )
```

Read the unique sensor ID of the HS3xxx.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
<i>id</i>	Pointer buffer to store the ID

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

5.7 Hardware Abstraction Layer API

Hardware abstraction layer (HAL) API, used by Renesas Environmental Sensor APIs to physically access the sensor.

Modules

- [ComBoard HAL](#)
Renesas communication board HAL.
- [Raspberry Pi HAL](#)
Renesas communication board HAL.
- [Custom HAL Template](#)
Template file for customer specific HAL implementation.

Files

- file [zmod4xxx_hal.h](#)
ZMOD4xxx specific hardware abstraction layer definitions.
- file [hal.h](#)
Renesas Environmental Sensor HAL definitions.

Functions

- int [zmod4xxx_init](#) ([zmod4xxx_dev_t](#) *dev, [Interface_t](#) *hal)
- int [HAL_Init](#) ([Interface_t](#) *hal)
Initialize hardware and populate [Interface_t](#) object.
- int [HAL_Deinit](#) ([Interface_t](#) *hal)
Cleanup before program exit.
- void [HAL_HandleError](#) (int errorCode, void const *context)
Error handling function.
- int [HAL_SetError](#) (int error, int scope, [ErrorStringGenerator_t](#) errStrFn)
Function storing error information.
- char const * [HAL_GetErrorInfo](#) (int *error, int *scope, char *str, int bufSize)
Get detailed information for last error.
- char const * [HAL_GetErrorString](#) (int error, int scope, char *str, int bufSize)
Error string generator for HAL-scoped errors.

Data Structures

- struct [Interface_t](#)
A structure of pointers to hardware specific functions. [More...](#)

Typedefs

- typedef int(* [I2CImpl_t](#)) (void *, [uint8_t](#), [uint8_t](#) *, int, [uint8_t](#) *, int)
Function pointer type defining signature of I2C functions.
- typedef char const *(* [ErrorStringGenerator_t](#)) (int, int, char *, int)
Function type used for generation of error strings.

Enumerations

- enum `GenericError_t` { `ecSuccess` = 0, `ecHLError` = 0x100 }
Error code definitions.
- enum `ErrorScope_t` { `esSensor` = 0x0000, `esAlgorithm` = 0x1000, `esInterface` = 0x2000, `esHAL` = 0x3000 }
Success status code and error scopes.
- enum `HALError_t` {
 `heNoInterface` = 1, `heNotImplemented`, `heI2CReadMissing`, `heI2CWriteMissing`,
 `heSleepMissing`, `heResetMissing` }
HAL scope error definitions.

5.7.1 Detailed Description

Hardware abstraction layer (HAL) API, used by Renesas Environmental Sensor APIs to physically access the sensor.

The HAL API is a generic API that is used by all Renesas types of Environmental Sensors. It defines the interface that the sensor API can use to access the sensor on an arbitrary hardware platform. Renesas provides HAL implementations for EVK boards, Raspberry PI and Arduino. In order to use a specific sensor API in the customer application, the customer must implement the HAL API for his hardware platform. For that purpose, the template file `template.c` is provided, which must be adapted to work on the customer hardware. Please refer to the comments in the template file and the documentation of the HAL API for details.

All HAL API related files are located in path `src/hal` and its subdirectories.

5.7.2 Data Structure Documentation

5.7.2.1 struct `Interface_t`

A structure of pointers to hardware specific functions.

Data Fields

- void * `handle`
- `I2CImpl_t` `i2cRead`
- `I2CImpl_t` `i2cWrite`
- void(* `msSleep`)(uint32_t ms)
- int(* `reset`)(void *`handle`)

5.7.2.1.1 Field Documentation

5.7.2.1.1.1 `handle`

`void* handle`

handle to physical interface

5.7.2.1.1.2 i2cRead

`I2CImpl_t i2cRead`

Pointer to I2C read implementation

The read operation may be preceded by a write to the same slave address. The function accepts a pointer to the interface handle, the slave address of the device to communicate with and two pairs of buffer pointer and buffer length. The first pair of buffer pointer / buffer length defines the data to be written, the second pair provides the pointer to the buffer where received data is stored and how many bytes shall be read. If the length field of the first buffer is not zero, the implementation must send an I2C write condition on the bus, transfer the data and send a repeated start condition followed by the corresponding read condition on the bus. If the length parameter of the first buffer is zero, no write is generated and the read is started immediately. The transmission must be terminated with a stop condition on the bus.

5.7.2.1.1.3 i2cWrite

`I2CImpl_t i2cWrite`

Pointer to I2C write implementation

For convenience, the write operation accepts two pairs of buffer pointer and buffer length. This allows to transfer addresses or commands prior to data without the need to manually concatenate transmit data in a buffer. The implementation of this function must generate a start condition followed by the I2C slave address of the target device, followed by all data from both buffers. At the end of the transmission a stop bit must be generated on the bus.

5.7.2.1.1.4 msSleep

`void( * msSleep) (uint32_t ms)`

Pointer to delay function

An implementation must delay execution by the specified number of ms

5.7.2.1.1.5 reset

`int( * reset) (void *handle)`

Pointer to reset function

Implementation must pulse the reset pin

5.7.3 Typedef Documentation

5.7.3.1 ErrorStringGenerator_t

`typedef char const*( * ErrorStringGenerator_t) (int, int, char *, int)`

Function type used for generation of error strings.

Functions of this type may be passed to [HAL_SetError\(\)](#) to generate meaningful descriptions of error conditions.

5.7.3.2 I2CImpl_t

```
typedef int( * I2CImpl_t) (void *, uint8_t, uint8_t *, int, uint8_t *, int)
```

Function pointer type defining signature of I2C functions.

This function pointer typedef is used in [Interface_t](#) objects to hold pointers to I2C implementations of read and write.

5.7.4 Enumeration Type Documentation

5.7.4.1 ErrorScope_t

```
enum ErrorScope_t
```

Success status code and error scopes.

Enumerator

esSensor	Sensor scope
esAlgorithm	Algorithm scope
esInterface	Interface scope
esHAL	HAL scope

5.7.4.2 GenericError_t

```
enum GenericError_t
```

Error code definitions.

Enumerator

ecSuccess	Operation completed successfully
ecHALError	Returned by sensor API if a HAL function failed. Specific information about the error can be obtained using the function HAL_GetErrorInfo() .

5.7.4.3 HALError_t

```
enum HALError_t
```

HAL scope error definitions.

When sensors are initialized (e.g. `init_hardware()`), the hal objects is checked whether all HAL functions required by the sensor are provided. If a function is missing one of the errors from this enumeration is returned.

Enumerator

heNoInterface	There was no interface found
heNotImplemented	The requested function is not implemented
heI2CReadMissing	Interface_t::i2cRead not provided
heI2CWriteMissing	Interface_t::i2cWrite not provided
heSleepMissing	Interface_t::msSleep not provided
heResetMissing	Interface_t::reset not provided

5.7.5 Function Documentation

5.7.5.1 HAL_Deinit()

```
int HAL_Deinit (  
    Interface\_t * hal )
```

Cleanup before program exit.

This function shall free up resources that have been allocated through [HAL_Init\(\)](#).

Parameters

<i>hal</i>	pointer to Interface_t object to be deinitialized
------------	---

Returns

error code

Return values

0	on success
!=0	in case of error

5.7.5.2 HAL_GetErrorInfo()

```
char const* HAL_GetErrorInfo (  
    int * error,
```

```
int * scope,
char * str,
int bufSize )
```

Get detailed information for last error.

Use this function in the error handler to obtain information about the last error code and which component generated it. In addition if an error string generator function was provided during error generation, this function may return a text string, describing the error in more detail.

Parameters

<i>error</i>	Pointer to integer, where the error code is written
<i>scope</i>	Pointer to integer, where the error scope (module that was generating the error is written)
<i>str</i>	Pointer to string buffer, where error message is written. If no string information is required, pass NULL pointer.
<i>bufSize</i>	Size of the string buffer, pass 0 if not used

Returns

Value passed in str

5.7.5.3 HAL_GetErrorString()

```
char const* HAL_GetErrorString (
    int error,
    int scope,
    char * str,
    int bufSize )
```

Error string generator for HAL-scoped errors.

This function generates error information for HAL scoped errors. Usually user code does not need to use this function directly.

Parameters

<i>error</i>	Error code for which error information is to be returned
<i>scope</i>	Error scope for which error information is to be returned
<i>str</i>	Pointer to string buffer, where error message is written
<i>bufSize</i>	Size of the string buffer

Returns

Value passed in str

5.7.5.4 HAL_HandleError()

```
void HAL_HandleError (
    int errorCode,
    void const * context )
```

Error handling function.

The implementation of this function defines the behavior of the application code when an error occurs during execution.

Parameters

<i>errorCode</i>	code of the error to be handled
<i>context</i>	additional context information

5.7.5.5 HAL_Init()

```
int HAL_Init (
    Interface_t * hal )
```

Initialize hardware and populate [Interface_t](#) object.

Any implementation must initialize those members of the [Interface_t](#) object that are required by the sensor being operated with pointers to functions that implement the behavior as specified in the [Interface_t](#) member documentation.

Parameters

<i>hal</i>	pointer to Interface_t object to be initialized
------------	---

Returns

error code

Return values

0	on success
!=0	in case of error

5.7.5.6 HAL_SetError()

```
int HAL_SetError (
    int error,
```

```
int scope,
ErrorStringGenerator_t errStrFn )
```

Function storing error information.

The sensor interface has different components which may generate error conditions. To keep the sensor interface as simple as possible, this function is called in case of an error. The return value of this function will be returned as error code of the function in which an error has occurred.

Internally, this function stores the error code and the scope of the error (that is which module was generating the error) in a data structure. For all errors which do not have the scope esSensor, this function will return the generic error code ecHALError. Error codes generated by the sensor are returned directly.

It is possible to pass an error string generator function along with the error information. If this function is provided, the error handler can query an error string, providing more meaningful error information.

Parameters

<i>error</i>	An error code
<i>scope</i>	The scope of the error (integer identifying a module)
<i>errStrFn</i>	Optional function pointer that can decode generate a meaningful message for the error code. Pass NULL if not used.

5.7.5.7 zmod4xxx_init()

```
int zmod4xxx_init (
    zmod4xxx_dev_t * dev,
    Interface_t * hal )
```

Find the sensor and initialize hal specific data

If example code is ported to the customer platform, this function must be re-implemented. The function must assign the `zmod4xxx_dev_t::read`, `zmod4xxx_dev_t::write` and `zmod4xxx_dev_t::delay_ms` members of `dev`.

Parameters

in	<i>dev</i>	pointer to the sensor object
in	<i>hal</i>	pointer to the hal interface object

Returns

error code

Return values

0	on success
!=0	hardware specific error code

5.8 ComBoard HAL

Renesas communication board HAL.

Modules

- [HiCom Board API](#)
- [ESCom Board API](#)

5.8.1 Detailed Description

Renesas communication board HAL.

The ComBoard HAL is used to provide access to Renesas Environmental Sensors through a communication board that is connected to the users PC. The ComBoard HAL supports both, the new ESCom board and the old HiCom board. Some sensors might not be working with the HiCom board, as this board does not support clock stretching.

The ComBoard HAL is implemented in file `src/hal/comboard/comboard.c`. The HAL functions use the [ESCom Board API](#) and [HiCom Board API](#) to discover connected communication boards and initialize the `Interface_t` hal object function pointers appropriately to operate the sensor.

5.9 Raspberry Pi HAL

Renesas communication board HAL.

Files

- file [rpi.h](#)
Raspberry Pi HAL type and function declarations.

Enumerations

- enum **RPiErrorDefs_t** { **resPiGPIO** = 0x310000, **resI2C** = 0x320000, **resI2CLenMismatch** = 0x320001 }

5.9.1 Detailed Description

Renesas communication board HAL.

The Raspberry Pi HAL is used to provide access to Renesas Environmental Sensors through the GPIO pins of a Raspberry Pi. The HAL functions are implemented in file `src/hal/raspi/rpi.c`.

5.10 Custom HAL Template

Template file for customer specific HAL implementation.

Template file for customer specific HAL implementation.

The file `src/hal/custom/template.c` serves as a starting point for customers to implement a HAL for their own target hardware. Refer to the file documentation for instructions on how to implement the required HAL functionality and use the implementations of the [Raspberry Pi HAL](#) or the [ComBoard HAL](#) as further reference.

5.11 ECom Board API

Files

- file [escom.h](#)
ECom board type and function declarations.

Functions

- int [ECom_Find](#) ([EComInterface_t](#) *boards, int *count)
- int [ECom_Connect](#) ([EComInterface_t](#) *board)
- int [ECom_Disconnect](#) ([EComInterface_t](#) *board)
- int [ECom_SetPower](#) ([EComInterface_t](#) *board, bool on)
- int [ECom_I2CWrite](#) ([EComInterface_t](#) *board, uint8_t slAddr, uint8_t *wData1, int wSize1, uint8_t *wData2, int wSize2)
- int [ECom_I2CRead](#) ([EComInterface_t](#) *board, uint8_t slAddr, uint8_t *wData, int wSize, uint8_t *rData, int rSize)
- int [ECom_GetSensorVoltage](#) ([EComInterface_t](#) *board, float *voltage)
- char const * [ECom_GetErrorString](#) (int error, int scope, char *buf, int bufLen)

Data Structures

- struct [EComInterface_t](#)

Enumerations

- enum [EComErrorScope_t](#) { **iesLibUSB** = 0x11, **iesI2C** = 0x12, **iesCommand** = 0x14 }
- enum **EComMode_t** { **emUSB**, **emEVK** }

5.11.1 Detailed Description

5.11.2 Data Structure Documentation

5.11.2.1 struct [EComInterface_t](#)

Data structure containing ECom board information

Data Fields

- libusb_device * **device**
- libusb_device_handle * **handle**
- uint8_t **outEP**
- uint8_t **inEP**
- wchar_t **serial** [33]

5.11.3 Enumeration Type Documentation

5.11.3.1 ESComErrorScope_t

enum `ESComErrorScope_t`

ESCom error scopes

These codes identify the component that generated an error code

5.11.4 Function Documentation

5.11.4.1 ESCom_Connect()

```
int ESCom_Connect (
    ESComInterface_t * board )
```

Connect ESCom board

This function tries to connect the ESCom board identified by `board` and allocates the required resources.

Parameters

in	<i>board</i>	Pointer to an ::ESComInterface_t instance obtained through ESCom_Find()
----	--------------	---

Returns

0 on success or error code on failure

5.11.4.2 ESCom_Disconnect()

```
int ESCom_Disconnect (
    ESComInterface_t * board )
```

Disconnect ESCom board and free associated resources

Parameters

in	<i>board</i>	Pointer to an ::ESComInterface_t instance
----	--------------	---

Returns

0 on success or error code on failure

5.11.4.3 ESCom_Find()

```
int ESCom_Find (
    ESComInterface_t * boards,
    int * count )
```

Enumerate ESCom boards connected to PC

Scan available USB interfaces and save those with matching VID/PID in the `boards` buffer. Value pointed to by `count` serves as input and output. As input, `count` specifies the size maximum number of boards to be stored in `boards`. The buffer must provide sufficient space. Before return, this function stores the number of detected ESCom boards in `count`.

Parameters

in	<i>boards</i>	Pointer to ESComInterface_t objects
in, out	<i>count</i>	Maximum number of boards to be stored in <code>board</code> [in], actual number of boards stored in <code>board</code> [out]

Returns

0 on success or error code on failure

5.11.4.4 ESCom_GetErrorString()

```
char const* ESCom_GetErrorString (
    int error,
    int scope,
    char * buf,
    int bufLen )
```

Generate a descriptive error message

Parameters

in	<i>error</i>	Error code
in	<i>scope</i>	Error scope

Returns

Error string

5.11.4.5 ESCom_GetSensorVoltage()

```
int ESCom_GetSensorVoltage (
    ESComInterface_t * board,
    float * voltage )
```

Read out the measured sensor voltage

Parameters

in	<i>board</i>	Pointer to an ::ESComInsteface_t instance
out	<i>voltage</i>	voltage value measured by ESCom board

5.11.4.6 ESCom_I2CRead()

```
int ESCom_I2CRead (
    ESComInterface_t * board,
    uint8_t sIAddr,
    uint8_t * wData,
    int wSize,
    uint8_t * rData,
    int rSize )
```

This is the ESCom implementation of HAL_t::i2cRead

5.11.4.7 ESCom_I2CWrite()

```
int ESCom_I2CWrite (
    ESComInterface_t * board,
    uint8_t sIAddr,
    uint8_t * wData1,
    int wSize1,
    uint8_t * wData2,
    int wSize2 )
```

This is the ESCom implementation of HAL_t::i2cWrite

5.11.4.8 ESCom_SetPower()

```
int ESCom_SetPower (
    ESComInterface_t * board,
    bool on )
```

Switch on or off the internal sensor supply voltage

Parameters

in	<i>board</i>	Pointer to an ::ESComInsteface_t instance obtained
in	<i>on</i>	Boolean indicating whether the sensor supply voltage shall be switched on

Returns

0 on success or error code otherwise

5.12 Return codes of the algorithm functions.

Macros

- #define ULP_O3_OK (0)
- #define ULP_O3_STABILIZATION (1)
- #define ULP_O3_DAMAGE (-102)

5.12.1 Detailed Description

5.12.2 Macro Definition Documentation

5.12.2.1 ULP_O3_DAMAGE

```
#define ULP_O3_DAMAGE (-102)
```

sensor damaged

5.12.2.2 ULP_O3_OK

```
#define ULP_O3_OK (0)
```

everything okay

5.12.2.3 ULP_O3_STABILIZATION

```
#define ULP_O3_STABILIZATION (1)
```

sensor in stabilization

Chapter 6

Data Structure Documentation

6.1 algorithm_version Struct Reference

Variables that describe the library version.

Data Fields

- uint8_t **major**
- uint8_t **minor**
- uint8_t **patch**

6.1.1 Detailed Description

Variables that describe the library version.

6.2 ulp_o3_handle_t Struct Reference

Variables that describe the sensor or the algorithm state.

Data Fields

- uint32_t **sample_cnt**
- float **smooth_rmx**
- float **gcda**
- float **o3_conc_ppb**
- float **o3_1h_ppb**
- float **o3_8h_ppb**

6.2.1 Detailed Description

Variables that describe the sensor or the algorithm state.

6.2.2 Field Documentation

6.2.2.1 gcda

float gcda

baseline conductance.

6.2.2.2 sample_cnt

uint32_t sample_cnt

Sample counter. Will saturate at 0xFFFFFFFF.

6.3 ulp_o3_inputs_t Struct Reference

Variables that are needed for algorithm.

Data Fields

- uint8_t * **adc_result**
- float **humidity_pct**
- float **temperature_degc**

6.3.1 Detailed Description

Variables that are needed for algorithm.

Parameters

in	<i>adc_result</i>	Value from read_adc_result function
in	<i>humidity_pct</i>	relative ambient humidity (%)
in	<i>temperature_degc</i>	ambient temperature (degC)

6.4 ulp_o3_results_t Struct Reference

Variables that receive the algorithm outputs.

Data Fields

- float `rmox` [8]
- float `temperature`
- float `O3_conc_ppb`
- uint16_t `FAST_AQI`
- uint16_t `EPA_AQI`

6.4.1 Detailed Description

Variables that receive the algorithm outputs.

6.4.2 Field Documentation

6.4.2.1 EPA_AQI

```
uint16_t EPA_AQI
```

EPA_AQI stands for the Air Quality Index according to the EPA standard based on ozone.

6.4.2.2 FAST_AQI

```
uint16_t FAST_AQI
```

FAST_AQI stands for a 1-minute average of the Air Quality Index according to the EPA standard based on ozone

6.4.2.3 O3_conc_ppb

```
float O3_conc_ppb
```

O3_conc_ppb stands for the ozone concentration in part-per-billion

6.4.2.4 rmox

```
float rmox[8]
```

MOx resistance.

6.4.2.5 temperature

```
float temperature
```

Temperature (degC) used for ambient compensation

Chapter 7

File Documentation

7.1 algos/ulp_o3.h File Reference

This file contains the data structure definitions and the function definitions for the ULP O3 algorithm.

Data Structures

- struct [algorithm_version](#)
Variables that describe the library version.
- struct [ulp_o3_handle_t](#)
Variables that describe the sensor or the algorithm state.
- struct [ulp_o3_results_t](#)
Variables that receive the algorithm outputs.
- struct [ulp_o3_inputs_t](#)
Variables that are needed for algorithm.

Macros

- #define [ULP_O3_OK](#) (0)
- #define [ULP_O3_STABILIZATION](#) (1)
- #define [ULP_O3_DAMAGE](#) (-102)

Functions

- [int8_t init_ulp_o3](#) ([ulp_o3_handle_t](#) *handle)
Initializes the ULP O3 algorithm.
- [int8_t calc_ulp_o3](#) ([ulp_o3_handle_t](#) *handle, [zmod4xxx_dev_t](#) *dev, const [ulp_o3_inputs_t](#) *algo_input, [ulp_o3_results_t](#) *results)
calculates results from present sample.

7.1.1 Function Documentation

7.1.1.1 calc_ulp_o3()

```
int8_t calc_ulp_o3 (
    ulp_o3_handle_t * handle,
    zmod4xxx_dev_t * dev,
    const ulp_o3_inputs_t * algo_input,
    ulp_o3_results_t * results )
```

calculates results from present sample.

Parameters

in	<i>handle</i>	Pointer to algorithm state variable.
in	<i>dev</i>	Pointer to the device.
in	<i>algo_input</i>	Structure containing inputs required for algo calculation.
out	<i>results</i>	Pointer for storing the algorithm results.

Returns

error code.

7.1.1.2 init_ulp_o3()

```
int8_t init_ulp_o3 (
    ulp_o3_handle_t * handle )
```

Initializes the ULP O3 algorithm.

Parameters

out	<i>handle</i>	Pointer to algorithm state variable.
-----	---------------	--------------------------------------

Returns

error code.

7.2 algos/zmod4510_config_ulp_o3.h File Reference

This is the configuration for ZMOD4510 module - ulp_o3 library.

Macros

- #define **INIT** 0
- #define **MEASUREMENT** 1
- #define **ZMOD4510_PID** 0x6320
- #define **ZMOD4510_I2C_ADDR** 0x33
- #define **ZMOD4510_PROD_DATA_LEN** 10
- #define **ZMOD4510_ADC_DATA_LEN** 18
- #define **ZMOD4510_ULP_O3_SAMPLE_TIME** (2000U)
- #define **ZMOD4XXX_H_ADDR** 0x40
- #define **ZMOD4XXX_D_ADDR** 0x50
- #define **ZMOD4XXX_M_ADDR** 0x60
- #define **ZMOD4XXX_S_ADDR** 0x68

Variables

- uint8_t **data_set_4510_init** []
- uint8_t **data_set_4510_ulp_o3** []
- [zmod4xxx_conf](#) **zmod_ulp_o3_sensor_cfg** []

7.2.1 Variable Documentation

7.2.1.1 data_set_4510_init

```
uint8_t data_set_4510_init[]
```

Initial value:

```
= {
    0x00, 0x50,
    0x00, 0x28, 0xC3, 0xE3,
    0x00, 0x00, 0x80, 0x40}
```

7.2.1.2 data_set_4510_ulp_o3

```
uint8_t data_set_4510_ulp_o3[]
```

Initial value:

```
= {
    0x00, 0x50, 0xFE, 0x70,
    0x00, 0x10,
    0x23, 0x03,
    0x00, 0x00, 0x06, 0x41,
    0x06, 0x41, 0x06, 0x41,
    0x06, 0x41, 0x06, 0x41,
    0x06, 0x41, 0x06, 0x41,
    0x86, 0x41
}
```

7.2.1.3 zmod_ulp_o3_sensor_cfg

zmod4xxx_conf zmod_ulp_o3_sensor_cfg[]

Initial value:

```
= {
    [INIT] = {
        .start = 0x80,
        .h = { .addr = ZMOD4XXX_H_ADDR, .len = 2, .data_buf = &data_set_4510_init[0]},
        .d = { .addr = ZMOD4XXX_D_ADDR, .len = 2, .data_buf = &data_set_4510_init[2]},
        .m = { .addr = ZMOD4XXX_M_ADDR, .len = 2, .data_buf = &data_set_4510_init[4]},
        .s = { .addr = ZMOD4XXX_S_ADDR, .len = 4, .data_buf = &data_set_4510_init[6]},
        .r = { .addr = 0x97, .len = 4},
    },

    [MEASUREMENT] = {
        .start = 0x80,
        .h = { .addr = ZMOD4XXX_H_ADDR, .len = 4, .data_buf = &data_set_4510_ulp_o3[0]},
        .d = { .addr = ZMOD4XXX_D_ADDR, .len = 2, .data_buf = &data_set_4510_ulp_o3[4]},
        .m = { .addr = ZMOD4XXX_M_ADDR, .len = 2, .data_buf = &data_set_4510_ulp_o3[6]},
        .s = { .addr = ZMOD4XXX_S_ADDR, .len = 18, .data_buf = &data_set_4510_ulp_o3[8]},
        .r = { .addr = 0x97, .len = 18},
        .prod_data_len = ZMOD4510_PROD_DATA_LEN,
    },
}
```

7.3 algos/zmod4xxx_cleaning.h File Reference

This file contains the cleaning function definition for ZMOD4xxx.

Functions

- int8_t zmod4xxx_cleaning_run (zmod4xxx_dev_t *dev)
Start a cleaning procedure.

7.3.1 Function Documentation

7.3.1.1 zmod4xxx_cleaning_run()

```
int8_t zmod4xxx_cleaning_run (
    zmod4xxx_dev_t * dev )
```

Start a cleaning procedure.

Parameters

in	dev	pointer to the device
----	-----	-----------------------

Returns

Error code

Return values

0	Success
!= 0	Error

7.4 hal/comboard/escom.h File Reference

ESCom board type and function declarations.

Data Structures

- struct [ESComInterface_t](#)

Enumerations

- enum [ESComErrorScope_t](#) { **iesLibUSB** = 0x11, **iesI2C** = 0x12, **iesCommand** = 0x14 }
- enum **ESComMode_t** { **emUSB**, **emEVK** }

Functions

- int [ESCom_Find](#) ([ESComInterface_t](#) *boards, int *count)
- int [ESCom_Connect](#) ([ESComInterface_t](#) *board)
- int [ESCom_Disconnect](#) ([ESComInterface_t](#) *board)
- int [ESCom_SetPower](#) ([ESComInterface_t](#) *board, bool on)
- int [ESCom_I2CWrite](#) ([ESComInterface_t](#) *board, uint8_t slAddr, uint8_t *wData1, int wSize1, uint8_t *wData2, int wSize2)
- int [ESCom_I2CRead](#) ([ESComInterface_t](#) *board, uint8_t slAddr, uint8_t *wData, int wSize, uint8_t *rData, int rSize)
- int [ESCom_GetSensorVoltage](#) ([ESComInterface_t](#) *board, float *voltage)
- char const * [ESCom_GetErrorString](#) (int error, int scope, char *buf, int bufLen)

7.5 hal/comboard/hicom.h File Reference

HiCom board type and function declarations.

Data Structures

- struct [HiComInterface_t](#)

Macros

- #define **HICOM_NAME** "Dual RS232-HS A"
- #define **HICOM_I2C_SPEED** 100000

Typedefs

- typedef FT_STATUS **HiComStatus_t**
- typedef FT_HANDLE **HiComHandle_t**

Enumerations

- enum **HiComErrorScope_t** { **iesFTDI** = 0x21 }

Functions

- int **HiCom_Find** (**HiComInterface_t** *board, int *count)
Enumerate HiCom boards Scans USB ports for connected HiCom boards.
- int **HiCom_Connect** (**HiComInterface_t** *board)
Connect a HiCom instance Tries to connect to an interface that has been discovered with [HiCom_Find\(\)](#) previously.
- int **HiCom_Disconnect** (**HiComInterface_t** *board)
Disconnect a HiComBoard.
- int **HiCom_SetPower** (**HiComInterface_t** *board, bool on)
Switch sensor power supply on or off.
- int **HiCom_I2CWrite** (**HiComInterface_t** *board, uint8_t slAddr, uint8_t *wData1, int wSize1, uint8_t *wData2, int wSize2)
- int **HiCom_I2CRead** (**HiComInterface_t** *board, uint8_t slAddr, uint8_t *wData, int wSize, uint8_t *rData, int rSize)
- char const * **HiCom_GetErrorString** (int error, int scope, char *buf, int bufLen)

7.6 hal/hal.h File Reference

Renesas Environmental Sensor HAL definitions.

Data Structures

- struct **Interface_t**
A structure of pointers to hardware specific functions. [More...](#)

Typedefs

- typedef int(* **I2CImpl_t**) (void *, uint8_t, uint8_t *, int, uint8_t *, int)
Function pointer type defining signature of I2C functions.
- typedef char const *(* **ErrorStringGenerator_t**) (int, int, char *, int)
Function type used for generation of error strings.

Enumerations

- enum `GenericError_t` { `ecSuccess` = 0, `ecHALError` = 0x100 }
Error code definitions.
- enum `ErrorScope_t` { `esSensor` = 0x0000, `esAlgorithm` = 0x1000, `esInterface` = 0x2000, `esHAL` = 0x3000 }
Success status code and error scopes.
- enum `HALError_t` {
 `heNoInterface` = 1, `heNotImplemented`, `heI2CReadMissing`, `heI2CWriteMissing`,
 `heSleepMissing`, `heResetMissing` }
HAL scope error definitions.

Functions

- int `HAL_Init` (`Interface_t` *hal)
Initialize hardware and populate `Interface_t` object.
- int `HAL_Deinit` (`Interface_t` *hal)
Cleanup before program exit.
- void `HAL_HandleError` (int errorCode, void const *context)
Error handling function.
- int `HAL_SetError` (int error, int scope, `ErrorStringGenerator_t` errStrFn)
Function storing error information.
- char const * `HAL_GetErrorInfo` (int *error, int *scope, char *str, int bufSize)
Get detailed information for last error.
- char const * `HAL_GetErrorString` (int error, int scope, char *str, int bufSize)
Error string generator for HAL-scoped errors.

7.7 hal/raspi/rpi.h File Reference

Raspberry PI HAL type and function declarations.

Enumerations

- enum `RPiErrorDefs_t` { `resPiGPIO` = 0x310000, `resI2C` = 0x320000, `recl2CLenMismatch` = 0x320001 }

7.8 hal/zmod4xxx_hal.h File Reference

ZMOD4xxx specific hardware abstraction layer definitions.

Functions

- int `zmod4xxx_init` (`zmod4xxx_dev_t` *dev, `Interface_t` *hal)

7.9 sensors/hs3xxx.h File Reference

HS3xxx sensor declarations.

Enumerations

- enum `HS3xxx_ErrorCodes_t` { `hteStaleData` = 1 }
Error code definitions specific for the HS3xxx API.

Functions

- int `HS3xxx_Init` (`HSxxxx_t` *sensor, `Interface_t` *hal)
Initialize the sensor object.
- int `HS3xxx_ReadID` (`HSxxxx_t` *sensor, uint32_t *id)
Read the unique sensor ID of the HS3xxx.
- int `HS3xxx_Measure` (`HSxxxx_t` *sensor, `HSxxxx_Results_t` *results)
Execute a temperature/humidity measurement cycle.
- int `HS3xxx_MeasureStart` (`HSxxxx_t` *sensor)
Start a temperature/humidity measurement cycle.
- int `HS3xxx_MeasureRead` (`HSxxxx_t` *sensor, `HSxxxx_Results_t` *results)
Read temperature/humidity results.

7.10 sensors/hs4xxx.h File Reference

HS4xxx sensor declarations.

Enumerations

- enum `HS4xxx_ErrorCodes_t` { `hteHS4xxxCRCError` = 1 }
Error code definitions specific for the HS4xxx API.

Functions

- int `HS4xxx_Init` (`HSxxxx_t` *sensor, `Interface_t` *hal)
Initialize the sensor object.
- int `HS4xxx_ReadID` (`HSxxxx_t` *sensor, uint32_t *id)
Read the unique sensor ID of the HS4xxx.
- int `HS4xxx_Measure` (`HSxxxx_t` *sensor, `HSxxxx_Results_t` *results)
Perform a temperature/humidity measurement cycle.
- int `HS4xxx_MeasureHold` (`HSxxxx_t` *sensor, `HSxxxx_Results_t` *results)
Perform a temperature/humidity measurement cycle using hold mode.
- int `HS4xxx_MeasureStart` (`HSxxxx_t` *sensor)
Start a temperature/humidity measurement cycle in non-hold mode.
- int `HS4xxx_MeasureRead` (`HSxxxx_t` *sensor, `HSxxxx_Results_t` *results)
Read temperature/humidity results in non-hold mode.

7.11 sensors/hsxxxx.h File Reference

Renesas humidity sensors (HS3xxx, HS4xxx) abstraction.

Data Structures

- struct [HSxxxx_Results_t](#)
Data structure holding humidity/temperature results. [More...](#)
- struct [HSxxxx_t](#)
Data structure holding information required for HSxxxx API operation. [More...](#)

Functions

- int [HSxxxx_Init](#) ([HSxxxx_t](#) *sensor, [Interface_t](#) *hal)
Initialize the sensor object.
- int [HSxxxx_Measure](#) ([HSxxxx_t](#) *sensor, [HSxxxx_Results_t](#) *results)
Perform one temperature measurement.
- char const * [HSxxxx_Name](#) ([HSxxxx_t](#) *sensor)
Return the temperature/humidity sensor type name.

7.12 sensors/zmod4xxx.h File Reference

zmod4xxx-API functions

Macros

- #define [ZMOD4XXX_ADDR_PID](#) (0x00)
- #define [ZMOD4XXX_ADDR_CONF](#) (0x20)
- #define [ZMOD4XXX_ADDR_PROD_DATA](#) (0x26)
- #define [ZMOD4XXX_ADDR_CMD](#) (0x93)
- #define [ZMOD4XXX_ADDR_STATUS](#) (0x94)
- #define [ZMOD4XXX_ADDR_TRACKING](#) (0x3A)
- #define [ZMOD4XXX_LEN_PID](#) (2)
- #define [ZMOD4XXX_LEN_CONF](#) (6)
- #define [ZMOD4XXX_LEN_TRACKING](#) (6)
- #define [HSP_MAX](#) (8)
- #define [RSLT_MAX](#) (32)
- #define [STATUS_SEQUENCER_RUNNING_MASK](#) (0x80)
- #define [STATUS_SLEEP_TIMER_ENABLED_MASK](#) (0x40)
- #define [STATUS_ALARM_MASK](#) (0x20)
- #define [STATUS_LAST_SEQ_STEP_MASK](#) (0x1F)
- #define [STATUS_POR_EVENT_MASK](#) (0x80)
- #define [STATUS_ACCESS_CONFLICT_MASK](#) (0x40)

Functions

- `zmod4xxx_err zmod4xxx_calc_factor (zmod4xxx_conf *conf, uint8_t *hsp, uint8_t *config)`
Calculate measurement settings.
- `float zmod4xxx_calc_single_rmx (zmod4xxx_dev_t *dev, uint8_t *adc_result)`
Calculate rmx resistance from ADC raw data.
- `zmod4xxx_err zmod4xxx_calc_rmx (zmod4xxx_dev_t *dev, uint8_t *adc_result, float *rmx)`
Calculate rmx resistance on array of results.
- `zmod4xxx_err zmod4xxx_check_error_event (zmod4xxx_dev_t *dev)`
Check the error event of the device.
- `zmod4xxx_err zmod4xxx_init_measurement (zmod4xxx_dev_t *dev)`
Initialize the sensor for corresponding measurement.
- `zmod4xxx_err zmod4xxx_init_sensor (zmod4xxx_dev_t *dev)`
Initialize the sensor after power on.
- `zmod4xxx_err zmod4xxx_null_ptr_check (zmod4xxx_dev_t *dev)`
Check if all function pointers are assigned.
- `zmod4xxx_err zmod4xxx_prepare_sensor (zmod4xxx_dev_t *dev)`
High-level function to prepare sensor.
- `zmod4xxx_err zmod4xxx_read_adc_result (zmod4xxx_dev_t *dev, uint8_t *adc_result)`
Read adc values from the sensor.
- `zmod4xxx_err zmod4xxx_read_rmx (zmod4xxx_dev_t *dev, uint8_t *adc_result, float *rmx)`
High-level function to read rmx.
- `zmod4xxx_err zmod4xxx_read_sensor_info (zmod4xxx_dev_t *dev)`
Read sensor parameter.
- `zmod4xxx_err zmod4xxx_read_status (zmod4xxx_dev_t *dev, uint8_t *status)`
Read the status of the device.
- `zmod4xxx_err zmod4xxx_read_tracking_number (zmod4xxx_dev_t *dev, uint8_t *track_num)`
Read tracking number of sensor.
- `zmod4xxx_err zmod4xxx_start_measurement (zmod4xxx_dev_t *dev)`
Start the measurement.
- `zmod4xxx_err zmod4xxx_start_measurement_at (zmod4xxx_dev_t *dev, uint8_t step)`
Start the measurement at an user-defined sequencer step.

7.13 sensors/zmod4xxx_types.h File Reference

zmod4xxx types

Data Structures

- struct `zmod4xxx_conf_str`
A single data set for the configuration. [More...](#)
- struct `zmod4xxx_conf`
Structure to hold the gas sensor module configuration. [More...](#)
- struct `zmod4xxx_dev_t`
Device structure ZMOD4xxx. [More...](#)

Typedefs

- typedef int8_t(* [zmod4xxx_i2c_ptr_t](#)) (uint8_t addr, uint8_t reg_addr, uint8_t *data_buf, uint8_t len)
function pointer type for i2c access
- typedef void(* [zmod4xxx_delay_ptr_p](#)) (uint32_t ms)
function pointer to hardware dependent delay function

Enumerations

- enum [zmod4xxx_err](#) {
 ZMOD4XXX_OK = 0, [ERROR_INIT_OUT_OF_RANGE](#), [ERROR_GAS_TIMEOUT](#), [ERROR_I2C](#) = -3,
 [ERROR_SENSOR_UNSUPPORTED](#), [ERROR_CONFIG_MISSING](#), [ERROR_ACCESS_CONFLICT](#), [ERROR_POR_EVENT](#),
 [ERROR_CLEANING](#), [ERROR_NULL_PTR](#) }
error_codes Error codes

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES ("RENESAS") PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01 Jan 2024)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.