ZMOD4xxx-API Documentation

API Version: 2.6.0

Contents

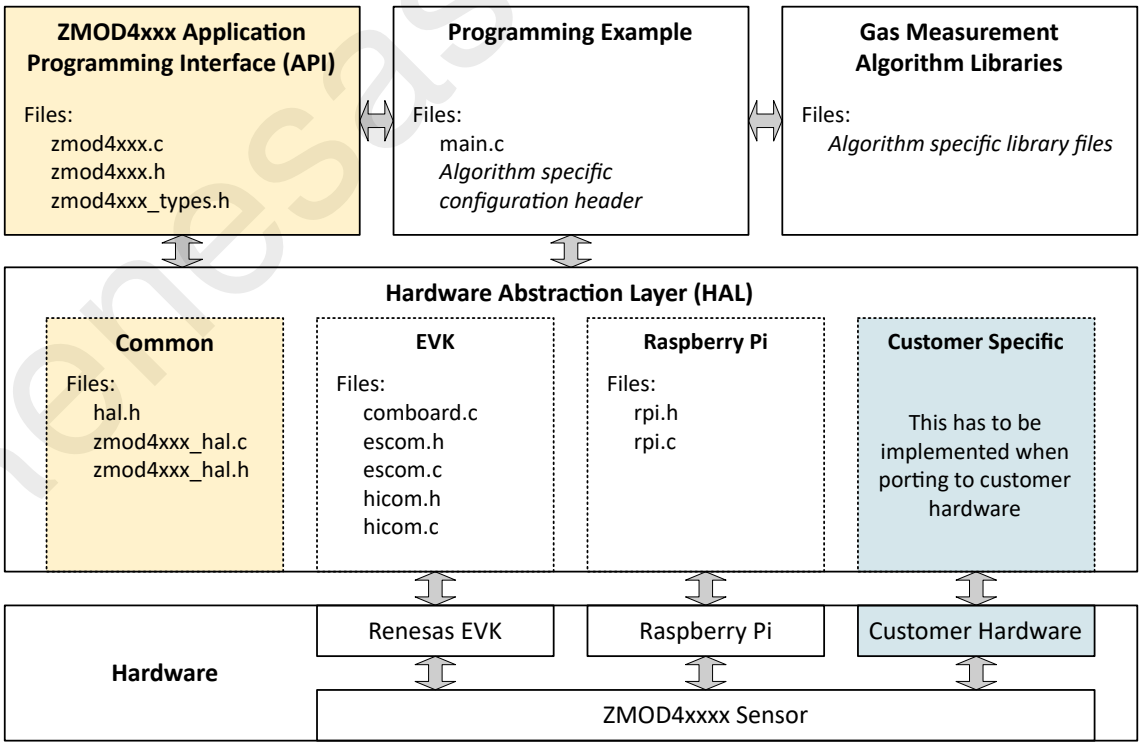
Chapter 1

ZMOD4xxx Application Programming Interface Overview

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.

The image below shows the source code structure of all ZMOD4xxx based programming examples. The document at hand provides documentation for the boxes with yellow background.



Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

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

Chapter 5

Module Documentation

5.1 ZMOD4xxx Sensor API

Files

- file zmod4xxx.h

  zmod4xxx-API functions
- file zmod4xxx_types.h

  zmod4xxx types

Data Structures

- struct zmod4xxx_conf_str

  A single data set for the configuration.
- struct zmod4xxx_conf

  Structure to hold the gas sensor module configuration.
- struct zmod4xxx_dev_t

  Device structure ZMOD4xxx.

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

Functions

- zmod4xxx_err zmod4xxx_calc_factor (zmod4xxx_conf ∗conf, uint8_t ∗hsp, uint8_t ∗config)

  Calculate measurement settings.
- zmod4xxx_err zmod4xxx_calc_rmox (zmod4xxx_dev_t ∗dev, uint8_t ∗adc_result, float ∗rmox)

  Calculate mox resistance.
- 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_rmox (zmod4xxx_dev_t ∗dev, uint8_t ∗adc_result, float ∗rmox)

  High-level function to read rmox.
- 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.

5.1.1 Detailed Description

Functions and types defined in this module are used to control the gas sensor and read raw measurement results. The sensor data obtained through this API is raw data that needs to be processed further by Renesas Gas Algorithms, provided in separate libraries.

The ZMOD4xxx API relies on an implementation of the Hardware Abstraction Layer API, to be able to communicate over I2C in the customer hardware.

5.1.2 Macro Definition Documentation

5.1.2.1 STATUS_ACCESS_CONFLICT_MASK

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

AccessConflict

5.1.2.2 STATUS_ALARM_MASK

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

Alarm

5.1.2.3 STATUS_LAST_SEQ_STEP_MASK

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

Last executed sequencer step

5.1.2.4 STATUS_POR_EVENT_MASK

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

POR_event

5.1.2.5 STATUS_SEQUENCER_RUNNING_MASK

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

Sequencer is running

5.1.2.6 STATUS_SLEEP_TIMER_ENABLED_MASK

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

SleepTimer_enabled

5.1.3 Typedef Documentation

5.1.3.1 zmod4xxx_delay_ptr_p

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

function pointer to hardware dependent delay function

Parameters

in	*delay*	in milliseconds

Returns

none

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

Returns

error code

Return values

0	success
!= 0	error

5.1.4 Enumeration Type Documentation

5.1.4.1 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.

5.1.5 Function Documentation

5.1.5.1 zmod4xxx_calc_factor()

```
zmod4xxx_err zmod4xxx_calc_factor (
            zmod4xxx_conf * conf,
            uint8_t * hsp,
            uint8_t * config )
```

Calculate measurement settings.

Parameters

in	*conf*	measurement configuration data
in	*hsp*	heater set point pointer
in	*config*	sensor configuration data pointer

Returns

error code

Return values

0	success

5.1.5.2 zmod4xxx_calc_rmox()

```
zmod4xxx_err zmod4xxx_calc_rmox (
             zmod4xxx_dev_t * dev,
             uint8_t * adc_result,
             float * rmox )
```

Calculate mox resistance.

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	*rmox*	pointer to the rmox values

Returns

error code

Return values

0	success
!= 0	error

5.1.5.3 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

5.1.5.4 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

5.1.5.5 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

5.1.5.6 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

5.1.5.7 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

5.1.5.8 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

5.1.5.9 zmod4xxx_read_rmox()

```
zmod4xxx_err zmod4xxx_read_rmox (
             zmod4xxx_dev_t * dev,
             uint8_t * adc_result,
             float * rmox )
```

High-level function to read rmox.

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	*rmox*	pointer to the rmox values

Returns

error code

Return values

0	success
!= 0	error

5.1.5.10 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.

5.1.5.11 zmod4xxx_read_status()

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

Read the status of the device.

Parameters

`in`	*dev*	pointer to the device
`in,out`	*status*	pointer to the status variable

Returns

error code

Return values

0	success
!= 0	error

5.1.5.12 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

in	*dev*	pointer to the device
in,out	*track_num*	pointer to buffer to store the tracking number

Returns

error code

Return values

0	success
!= 0	error

5.1.5.13 zmod4xxx_start_measurement()

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

Start the measurement.

Parameters

in	*dev*	pointer to the device

Returns

error code

Return values

0	success
!= 0	error

5.2 Hardware Abstraction Layer API

Files

- file hal.h

  Generic hardware abstraction layer definitions.
- file zmod4xxx_hal.h

  ZMOD4xxx specific hardware abstraction layer definitions.

Data Structures

- struct HAL_t

  A structure of pointers to hardware specific functions.

Enumerations

- enum ErrorCommon_t {
  ecSuccess = 0, esSensor = 0x00000000, esAlgorithm = 0x10000000, esInterface = 0x20000000,
  esHAL = 0x30000000, esMask = 0xf0000000 }

  Success status code and error scopes.
- enum HALError_t {
  heI2CReadRequried = esHAL | 1, heI2CWriteRequried, heI2CWriteReadRequried, heSleepRequried,
  heResetRequried }

  HAL scope error definitions.
- enum InterfaceError_t { **ieNoInterfaceFound** = esInterface + 1 }

  Interface scope error definitions.

Functions

- int HAL_Init (HAL_t *hal)

  Initialize hardware and populate HAL_t object.
- int HAL_Deinit (HAL_t *hal)

  Cleanup before program exit.
- void HAL_HandleError (int errorCode, void const *context)

  Example error handler.
- int init_hardware (zmod4xxx_dev_t *dev)
- int deinit_hardware ()

5.2.1 Detailed Description

The Hardware Abstraction Layer (HAL) API provides a generic interface for I2C communication and other hardware specific functionality that may be required by a sensor. When code is ported to a new platform, these functions must be re-implemented.

Note

The HAL is designed to support different sensors. Typically a sensor does not require all functions provided by HAL_t.

5.2.2 Enumeration Type Documentation

5.2.2.1 ErrorCommon_t

`enum ErrorCommon_t`

Success status code and error scopes.

Enumerator

ecSuccess	common success code
esSensor	Sensor scope
esAlgorithm	Algorithm scope
esInterface	Interface scope
esHAL	HAL scope
esMask	provided for scope filtering

5.2.2.2 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

heI2CReadRequried	HAL_t::i2cRead not provided
heI2CWriteRequried	HAL_t::i2cWrite not provided
heI2CWriteReadRequried	HAL_t::i2cWriteRead not provided
heSleepRequried	HAL_t::sleepMs not provided
heResetRequried	HAL_t::reset not provided

5.2.3 Function Documentation

5.2.3.1 deinit_hardware()

`int deinit_hardware ( )`

Free up resources allocated by init_hardware

Returns

error code

Return values

0	on success
!=0	hardware specific error code

5.2.3.2 HAL_Deinit()

```
int HAL_Deinit (
            HAL_t * hal )
```

Cleanup before program exit.

This function shall free up resources that have been allocated through HAL_Init().

Parameters

hal	pointer to HAL_t object to be deinitialized

Returns

error code

Return values

0	on success
!=0	in case of error

5.2.3.3 HAL_HandleError()

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

Example error handler.

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

Parameters

errorCode	code of the error to be handled
context	additional context information

5.2.3.4 HAL_Init()

```
int HAL_Init (
              HAL_t * hal )
```

Initialize hardware and populate HAL_t object.

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

Parameters

hal	pointer to HAL_t object to be initialized

Returns

error code

Return values

0	on success
!=0	in case of error

5.2.3.5 init_hardware()

```
int init_hardware (
          zmod4xxx_dev_t * dev )
```

Init hardware and assign hardware specific functions to ZMOD4xxx object

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	*dev*	pointer to the sensor object

Returns

error code

Return values

0	on success
!=0	hardware specific error code

Chapter 6

Data Structure Documentation

6.1 HAL_t Struct Reference

A structure of pointers to hardware specific functions.

`#include <hal.h>`

Data Fields

- int(∗ i2cRead)(uint8_t slAddr, uint8_t ∗rdData, int rdSize)
- int(∗ i2cWrite)(uint8_t slAddr, uint8_t ∗wrData, int wrSize)
- int(∗ i2cWriteRead)(uint8_t slAddr, uint8_t ∗wrData, int wrSize, uint8_t ∗rdData, int rdSize)
- void(∗ msSleep)(uint32_t ms)
- int(∗ reset)()

6.1.1 Detailed Description

A structure of pointers to hardware specific functions.

6.1.2 Field Documentation

6.1.2.1 i2cRead

```
int( * i2cRead) (uint8_t slAddr, uint8_t *rdData, int rdSize)
```

Pointer to I2C read implementation

An implementation must

- Send start bit
- Send (`slAddr` << 1) | 1
- Read `rdSize` bytes into `rdData`
- Send stop bit

6.1.2.2 i2cWrite

```
int( * i2cWrite) (uint8_t slAddr, uint8_t *wrData, int wrSize)
```

Pointer to I2C write implementation

An implementation must

- Send start bit
- Send (`slAddr` << 1)
- Send `wrSize` bytes (from `wrData`)
- Send stop bit

6.1.2.3 i2cWriteRead

```
int( * i2cWriteRead) (uint8_t slAddr, uint8_t *wrData, int wrSize, uint8_t *rdData, int rdSize)
```

Pointer to I2C write & read implementation

An implementation must

- Send start bit
- Send (`slAddr` << 1)
- Send `wrSize` bytes (from `wrData`)
- Send start bit (repeated start)
- Send (`slAddr` << 1) | 1
- Read `rdSize` bytes into `rdData`
- Send stop bit

6.1.2.4 msSleep

```
void( * msSleep) (uint32_t ms)
```

Pointer to delay function

An implementation must delay execution by the specified number of `ms`

6.1.2.5 reset

```
int( * reset) ()
```

Pointer to reset function

Implementation must pulse the reset pin

The documentation for this struct was generated from the following file:

- hal.h

6.2 zmod4xxx_conf Struct Reference

Structure to hold the gas sensor module configuration.

```
#include <zmod4xxx_types.h>
```

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**

6.2.1 Detailed Description

Structure to hold the gas sensor module configuration.

The documentation for this struct was generated from the following file:

- zmod4xxx_types.h

6.3 zmod4xxx_conf_str Struct Reference

A single data set for the configuration.

```
#include <zmod4xxx_types.h>
```

Data Fields

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

6.3.1 Detailed Description

A single data set for the configuration.

The documentation for this struct was generated from the following file:

- zmod4xxx_types.h

6.4 zmod4xxx_dev_t Struct Reference

Device structure ZMOD4xxx.

```
#include <zmod4xxx_types.h>
```

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

6.4.1 Detailed Description

Device structure ZMOD4xxx.

6.4.2 Field Documentation

6.4.2.1 config

```
uint8_t config[6]
```

configuration parameter set

6.4.2.2 delay_ms

```
zmod4xxx_delay_ptr_p delay_ms
```

function pointer to delay function

6.4.2.3 i2c_addr

```
uint8_t i2c_addr
```

i2c address of the sensor

6.4.2.4 init_conf

```
zmod4xxx_conf* init_conf
```

pointer to the init configuration

6.4.2.5 meas_conf

```
zmod4xxx_conf* meas_conf
```

pointer to the measurement configuration

6.4.2.6 mox_er

```
uint16_t mox_er
```

sensor specific parameter

6.4.2.7 mox_lr

```
uint16_t mox_lr
```

sensor specific parameter

6.4.2.8 pid

```
uint16_t pid
```

product id of the sensor

6.4.2.9 prod_data

```
uint8_t* prod_data
```

production data

6.4.2.10 read

```
zmod4xxx_i2c_ptr_t read
```

function pointer to i2c read

6.4.2.11 write

```
zmod4xxx_i2c_ptr_t write
```

function pointer to i2c write

The documentation for this struct was generated from the following file:

- zmod4xxx_types.h

Chapter 7

File Documentation

7.1 hal.h File Reference

Generic hardware abstraction layer definitions.

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

Data Structures

- struct HAL_t

  A structure of pointers to hardware specific functions.

Enumerations

- enum ErrorCommon_t {
  ecSuccess = 0, esSensor = 0x00000000, esAlgorithm = 0x10000000, esInterface = 0x20000000,
  esHAL = 0x30000000, esMask = 0xf0000000 }

  Success status code and error scopes.
- enum HALError_t {
  heI2CReadRequried = esHAL | 1, heI2CWriteRequried, heI2CWriteReadRequried, heSleepRequried,
  heResetRequried }

  HAL scope error definitions.
- enum InterfaceError_t { **ieNoInterfaceFound** = esInterface + 1 }

  Interface scope error definitions.

Functions

- int HAL_Init (HAL_t *hal)

  Initialize hardware and populate HAL_t object.
- int HAL_Deinit (HAL_t *hal)

  Cleanup before program exit.
- void HAL_HandleError (int errorCode, void const *context)

  Example error handler.

7.2 zmod4xxx.h File Reference

zmod4xxx-API functions

```
#include "zmod4xxx_types.h"
```

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.
- zmod4xxx_err zmod4xxx_calc_rmox (zmod4xxx_dev_t ∗dev, uint8_t ∗adc_result, float ∗rmox)

  Calculate mox resistance.
- 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_rmox (zmod4xxx_dev_t ∗dev, uint8_t ∗adc_result, float ∗rmox)

  High-level function to read rmox.
- 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.

7.3 zmod4xxx_hal.h File Reference

ZMOD4xxx specific hardware abstraction layer definitions.

```
#include "zmod4xxx_types.h"
```

Functions

- int init_hardware (zmod4xxx_dev_t *dev)
- int deinit_hardware ()

7.4 zmod4xxx_types.h File Reference

zmod4xxx types

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

Data Structures

- struct zmod4xxx_conf_str

  A single data set for the configuration.
- struct zmod4xxx_conf

  Structure to hold the gas sensor module configuration.
- struct zmod4xxx_dev_t

  Device structure ZMOD4xxx.

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

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 skilled in the art 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 for development of 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 out of 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 o any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.0 Mar 2020)

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/

Trademarks

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