PROJECT ETHER

Generated by Doxygen 1.11.0

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 bme280_compensator_t Struct Reference

Structure for the BME280 sensor compensator data.

`#include <bme280.h>`

Data Fields

- uint16_t dig_t1
- int16_t dig_t2
- int16_t dig_t3
- uint16_t dig_p1
- int16_t dig_p2
- int16_t dig_p3
- int16_t dig_p4
- int16_t dig_p5
- int16_t dig_p6
- int16_t dig_p7
- int16_t dig_p8
- int16_t dig_p9
- uint8_t dig_h1
- int16_t dig_h2
- uint8_t dig_h3
- int16_t dig_h4
- int16_t dig_h5
- int8_t dig_h6

3.1.1 Detailed Description

Structure for the BME280 sensor compensator data.

3.1.2 Field Documentation

3.1.2.1 dig_h1

```
uint8_t dig_h1
```

Humidity compensation value H1.

3.1.2.2 dig_h2

```
int16_t dig_h2
```

Humidity compensation value H2.

3.1.2.3 dig_h3

```
uint8_t dig_h3
```

Humidity compensation value H3.

3.1.2.4 dig_h4

```
int16_t dig_h4
```

Humidity compensation value H4.

3.1.2.5 dig_h5

```
int16_t dig_h5
```

Humidity compensation value H5.

3.1.2.6 dig_h6

```
int8_t dig_h6
```

Humidity compensation value H6.

3.1.2.7 dig_p1

```
uint16_t dig_p1
```

Pressure compensation value P1.

3.1.2.8 dig_p2

```
int16_t dig_p2
```

Pressure compensation value P2.

3.1.2.9 dig_p3

```
int16_t dig_p3
```

Pressure compensation value P3.

3.1.2.10 dig_p4

```
int16_t dig_p4
```

Pressure compensation value P4.

3.1.2.11 dig_p5

```
int16_t dig_p5
```

Pressure compensation value P5.

3.1.2.12 dig_p6

```
int16_t dig_p6
```

Pressure compensation value P6.

3.1.2.13 dig_p7

```
int16_t dig_p7
```

Pressure compensation value P7.

3.1.2.14 dig_p8

```
int16_t dig_p8
```

Pressure compensation value P8.

3.1.2.15 dig_p9

```
int16_t dig_p9
```

Pressure compensation value P9.

3.1.2.16 dig_t1

```
uint16_t dig_t1
```

Temperature compensation value T1.

3.1.2.17 dig_t2

```
int16_t dig_t2
```

Temperature compensation value T2.

3.1.2.18 dig_t3

```
int16_t dig_t3
```

Temperature compensation value T3.

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

- bme280.h

3.2 bme280_humidity_t Struct Reference

Structure for the BME280 sensor humidity data.

```
#include <bme280.h>
```

Data Fields

- uint8_t msb
- uint8_t lsb
- double compensated

3.2.1 Detailed Description

Structure for the BME280 sensor humidity data.

3.2.2 Field Documentation

3.2.2.1 compensated

```
double compensated
```

Compensated humidity value.

3.2.2.2 lsb

```
uint8_t lsb
```

Least significant byte.

3.2.2.3 msb

```
uint8_t msb
```

Most significant byte.

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

- bme280.h

3.3 bme280_measurements_t Struct Reference

Structure for BME280 sensor measurements.

```
#include <bme280.h>
```

Data Fields

- bme280_humidity_t humidity
- bme280_pressure_t pressure
- bme280_temperature_t temperature
- bme280_compensator_t compensator

3.3.1 Detailed Description

Structure for BME280 sensor measurements.

3.3.2 Field Documentation

3.3.2.1 compensator

```
bme280_compensator_t compensator
```

Compensator data.

3.3.2.2 humidity

```
bme280_humidity_t humidity
```

Humidity data.

3.3.2.3 pressure

```
bme280_pressure_t pressure
```

Pressure data.

3.3.2.4 temperature

```
bme280_temperature_t temperature
```

Temperature data.

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

- bme280.h

3.4 bme280_pressure_t Struct Reference

Structure for the BME280 sensor pressure data.

```
#include <bme280.h>
```

Data Fields

- uint8_t msb
- uint8_t lsb
- uint8_t xlsb
- double compensated

3.4.1 Detailed Description

Structure for the BME280 sensor pressure data.

3.4.2 Field Documentation

3.4.2.1 compensated

```
double compensated
```

Compensated pressure value.

3.4.2.2 lsb

```
uint8_t lsb
```

Least significant byte.

3.4.2.3 msb

```
uint8_t msb
```

Most significant byte.

3.4.2.4 xlsb

```
uint8_t xlsb
```

Extended least significant byte.

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

- bme280.h

3.5 bme280_settings_t Struct Reference

Structure for the BME280 sensor settings.

```
#include <bme280.h>
```

Data Fields

- uint8_t ctrl_hum
- uint8_t ctrl_meas
- uint8_t config

3.5.1 Detailed Description

Structure for the BME280 sensor settings.

3.5.2 Field Documentation

3.5.2.1 config

```
uint8_t config
```

Configuration register value.

3.5.2.2 ctrl_hum

```
uint8_t ctrl_hum
```

Control humidity register value.

3.5.2.3 ctrl_meas

```
uint8_t ctrl_meas
```

Control measurement register value.

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

- bme280.h

3.6 bme280_temperature_t Struct Reference

Structure for the BME280 sensor temperature data.

```
#include <bme280.h>
```

Data Fields

- uint8_t msb
- uint8_t lsb
- uint8_t xlsb
- double compensated

3.6.1 Detailed Description

Structure for the BME280 sensor temperature data.

3.6.2 Field Documentation

3.6.2.1 compensated

```
double compensated
```

Compensated temperature value.

3.6.2.2 lsb

```
uint8_t lsb
```

Least significant byte.

3.6.2.3 msb

```
uint8_t msb
```

Most significant byte.

3.6.2.4 xlsb

```
uint8_t xlsb
```

Extended least significant byte.

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

- bme280.h

3.7 ether_descriptor_t Struct Reference

Structure to hold various controller descriptors.

```
#include <ether.h>
```

Data Fields

- i2c_controller_descriptor_t i2c_controller
- mqtt_controller_descriptor_t mqtt_controller
- uart_controller_descriptor_t uart_controller
- wifi_controller_descriptor_t wifi_controller

3.7.1 Detailed Description

Structure to hold various controller descriptors.

3.7.2 Field Documentation

3.7.2.1 i2c_controller

```
i2c_controller_descriptor_t i2c_controller
```

I2C controller descriptor.

3.7.2.2 mqtt_controller

```
mqtt_controller_descriptor_t mqtt_controller
```

MQTT controller descriptor.

3.7.2.3 uart_controller

```
uart_controller_descriptor_t uart_controller
```

UART controller descriptor.

3.7.2.4 wifi_controller

```
wifi_controller_descriptor_t wifi_controller
```

WIFI controller descriptor.

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

- ether.h

3.8 ether_measurements_t Struct Reference

Structure to store PMS7003 and BME280 sensors measurements.

```
#include <ether.h>
```

Data Fields

- pms7003_measurements_t pms7003
- bme280_measurements_t bme280

3.8.1 Detailed Description

Structure to store PMS7003 and BME280 sensors measurements.

3.8.2 Field Documentation

3.8.2.1 bme280

```
bme280_measurements_t bme280
```

BME280 measurements.

3.8.2.2 pms7003

```
pms7003_measurements_t pms7003
```

PMS7003 measurements.

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

- ether.h

3.9 ether_settings_t Struct Reference

Structure for ETHER settings.

`#include <ether.h>`

Data Fields

- bme280_settings_t bme280

3.9.1 Detailed Description

Structure for ETHER settings.

3.9.2 Field Documentation

3.9.2.1 bme280

`bme280_settings_t bme280`

BME280 sensor settings.

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

- ether.h

3.10 ether_state_machine_t Struct Reference

Structure to hold the state machine for PMS7003 and BME280 sensors.

`#include <ether.h>`

Data Fields

- pms7003_state_t pms7003
- bme280_state_t bme280

3.10.1 Detailed Description

Structure to hold the state machine for PMS7003 and BME280 sensors.

3.10.2 Field Documentation

3.10.2.1 bme280

`bme280_state_t bme280`

BME280 sensor state.

3.10.2.2 pms7003

`pms7003_state_t pms7003`

PMS7003 sensor state.

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

- ether.h

3.11 ether_t Struct Reference

Main structure for ETHER containing measurements, descriptors, settings, and state machine.

`#include <ether.h>`

Data Fields

- ether_measurements_t measurements
- ether_descriptor_t descriptor
- ether_settings_t settings
- ether_state_machine_t state_machine

3.11.1 Detailed Description

Main structure for ETHER containing measurements, descriptors, settings, and state machine.

3.11.2 Field Documentation

3.11.2.1 descriptor

`ether_descriptor_t descriptor`

Controller descriptors.

3.11.2.2 measurements

```
ether_measurements_t measurements
```

Measurements data.

3.11.2.3 settings

```
ether_settings_t settings
```

Settings data.

3.11.2.4 state_machine

```
ether_state_machine_t state_machine
```

State machine data.

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

- ether.h

3.12 i2c_controller_descriptor_t Struct Reference

Structure for I2C controller descriptor.

```
#include <i2c_controller.h>
```

Data Fields

- i2c_config_t config
- i2c_port_t i2c_num

3.12.1 Detailed Description

Structure for I2C controller descriptor.

3.12.2 Field Documentation

3.12.2.1 config

```
i2c_config_t config
```

I2C configuration.

3.12.2.2 i2c_num

```
i2c_port_t i2c_num
```

I2C port number.

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

- i2c_controller.h

3.13 mqtt_controller_descriptor_t Struct Reference

Structure for MQTT controller descriptor.

```
#include <mqtt_controller.h>
```

Data Fields

- esp_mqtt_client_config_t client_config
- esp_mqtt_client_handle_t client_handle
- mqtt_controller_event_handler_t event_handler

3.13.1 Detailed Description

Structure for MQTT controller descriptor.

3.13.2 Field Documentation

3.13.2.1 client_config

```
esp_mqtt_client_config_t client_config
```

MQTT client configuration.

3.13.2.2 client_handle

```
esp_mqtt_client_handle_t client_handle
```

MQTT client handle.

3.13.2.3 event_handler

```
mqtt_controller_event_handler_t event_handler
```

MQTT event handler.

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

- mqtt_controller.h

3.14 pms7003_frame_answer_t Union Reference

Union representing a PMS7003 frame answer.

`#include <pms7003.h>`

Public Member Functions

- struct {
  uint8_t start_byte_1
  uint8_t start_byte_2
  uint16_t length
  uint16_t data_pm1_standard
  uint16_t data_pm25_standard
  uint16_t data_pm10_standard
  uint16_t data_pm1_atmospheric
  uint16_t data_pm25_atmospheric
  uint16_t data_concentration_unit
  uint16_t data_particles_300nm
  uint16_t data_particles_500nm
  uint16_t data_particles_1000nm
  uint16_t data_particles_2500nm
  uint16_t data_particles_5000nm
  uint16_t data_particles_10000nm
  uint16_t reserved
  uint16_t check_code
  } __attribute__ ((packed))

Data Fields

- uint8_t buffer_answer [PMS7003_FRAME_ANSWER_SIZE]

3.14.1 Detailed Description

Union representing a PMS7003 frame answer.

3.14.2 Member Function Documentation

3.14.2.1 __attribute__()

```
struct pms7003_frame_answer_t::@1 __attribute__ (
            (packed) )
```

3.14.3 Field Documentation

3.14.3.1 buffer_answer

```
uint8_t buffer_answer[PMS7003_FRAME_ANSWER_SIZE]
```

Buffer for answer frame.

3.14.3.2 check_code

```
uint16_t check_code
```

Checksum code.

3.14.3.3 data_concentration_unit

```
uint16_t data_concentration_unit
```

Concentration unit.

3.14.3.4 data_particles_10000nm

```
uint16_t data_particles_10000nm
```

Particles count for 10µm.

3.14.3.5 data_particles_1000nm

```
uint16_t data_particles_1000nm
```

Particles count for 1.0µm.

3.14.3.6 data_particles_2500nm

```
uint16_t data_particles_2500nm
```

Particles count for 2.5µm.

3.14.3.7 data_particles_300nm

```
uint16_t data_particles_300nm
```

Particles count for 0.3µm.

3.14.3.8 data_particles_5000nm

```
uint16_t data_particles_5000nm
```

Particles count for 5.0µm.

3.14.3.9 data_particles_500nm

```
uint16_t data_particles_500nm
```

Particles count for 0.5µm.

3.14.3.10 data_pm10_standard

```
uint16_t data_pm10_standard
```

PM10 concentration (standard particles).

3.14.3.11 data_pm1_atmospheric

```
uint16_t data_pm1_atmospheric
```

PM1.0 concentration (atmospheric particles).

3.14.3.12 data_pm1_standard

```
uint16_t data_pm1_standard
```

PM1.0 concentration (standard particles).

3.14.3.13 data_pm25_atmospheric

```
uint16_t data_pm25_atmospheric
```

PM2.5 concentration (atmospheric particles).

3.14.3.14 data_pm25_standard

```
uint16_t data_pm25_standard
```

PM2.5 concentration (standard particles).

3.14.3.15 length

```
uint16_t length
```

Length of the frame.

3.14.3.16 reserved

```
uint16_t reserved
```

Reserved bytes.

3.14.3.17 start_byte_1

```
uint8_t start_byte_1
```

Start byte 1.

3.14.3.18 start_byte_2

```
uint8_t start_byte_2
```

Start byte 2.

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

- pms7003.h

3.15 pms7003_frame_request_t Union Reference

Union representing a PMS7003 frame request.

```
#include <pms7003.h>
```

Public Member Functions

- struct {
  const uint8_t start_byte_1
  const uint8_t start_byte_2
  uint8_t command
  uint8_t data_h
  uint8_t data_l
  uint8_t lrch
  uint8_t lrcl
  } __attribute__ ((packed))

Data Fields

- uint8_t buffer_request [PMS7003_FRAME_REQUEST_SIZE]

3.15.1 Detailed Description

Union representing a PMS7003 frame request.

3.15.2 Member Function Documentation

3.15.2.1 __attribute__()

```
struct pms7003_frame_request_t::@0 __attribute__ (
            (packed) )
```

3.15.3 Field Documentation

3.15.3.1 buffer_request

```
uint8_t buffer_request[PMS7003_FRAME_REQUEST_SIZE]
```

Buffer for request frame.

3.15.3.2 command

```
uint8_t command
```

Command byte.

3.15.3.3 data_h

```
uint8_t data_h
```

Data high byte.

3.15.3.4 data_l

```
uint8_t data_l
```

Data low byte.

3.15.3.5 lrch

```
uint8_t lrch
```

High byte of the checksum.

3.15.3.6 lrcl

```
uint8_t lrcl
```

Low byte of the checksum.

3.15.3.7 start_byte_1

```
const uint8_t start_byte_1
```

Start byte 1.

3.15.3.8 start_byte_2

```
const uint8_t start_byte_2
```

Start byte 2.

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

- pms7003.h

3.16 pms7003_measurements_t Struct Reference

Structure representing PMS7003 measurements.

```
#include <pms7003.h>
```

Data Fields

- uint16_t pm1
- uint16_t pm25
- uint16_t pm10

3.16.1 Detailed Description

Structure representing PMS7003 measurements.

3.16.2 Field Documentation

3.16.2.1 pm1

```
uint16_t pm1
```

PM1.0 measurement.

3.16.2.2 pm10

```
uint16_t pm10
```

PM10 measurement.

3.16.2.3 pm25

```
uint16_t pm25
```

PM2.5 measurement.

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

- pms7003.h

3.17 uart_controller_descriptor_t Struct Reference

Structure for UART controller descriptor.

`#include <uart_controller.h>`

Data Fields

- uart_config_t uart_config
- uart_port_t uart_port

3.17.1 Detailed Description

Structure for UART controller descriptor.

3.17.2 Field Documentation

3.17.2.1 uart_config

`uart_config_t uart_config`

UART configuration.

3.17.2.2 uart_port

`uart_port_t uart_port`

UART port number.

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

- uart_controller.h

3.18 wifi_controller_descriptor_t Struct Reference

Structure for WIFI controller descriptor.

`#include <wifi_controller.h>`

Data Fields

- wifi_config_t wifi_config
- wifi_controller_event_handler_t event_handler

3.18.1 Detailed Description

Structure for WIFI controller descriptor.

3.18.2 Field Documentation

3.18.2.1 event_handler

```
wifi_controller_event_handler_t event_handler
```

WIFI event handler.

3.18.2.2 wifi_config

```
wifi_config_t wifi_config
```

WIFI configuration.

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

- wifi_controller.h

Chapter 4

File Documentation

4.1 bme280.h File Reference

```
#include <stddef.h>
#include <stdint.h>
#include "hal/i2c_types.h"
#include "i2c_controller.h"
```

Data Structures

- struct bme280_settings_t

  Structure for the BME280 sensor settings.
- struct bme280_pressure_t

  Structure for the BME280 sensor pressure data.
- struct bme280_temperature_t

  Structure for the BME280 sensor temperature data.
- struct bme280_humidity_t

  Structure for the BME280 sensor humidity data.
- struct bme280_compensator_t

  Structure for the BME280 sensor compensator data.
- struct bme280_measurements_t

  Structure for BME280 sensor measurements.

Macros

- #define BME280_I2C_ADDRESS (0x76)
- #define BME280_REGISTER_ID (0xd0)
- #define BME280_REGISTER_RESET (0xe0)
- #define BME280_REGISTER_CTRL_HUM (0xf2)
- #define BME280_REGISTER_STATUS (0xf3)
- #define BME280_REGISTER_CTRL_MEAS (0xf4)
- #define BME280_REGISTER_CONFIG (0xf5)
- #define BME280_REGISTER_PRESS_MSB (0xf7)
- #define BME280_REGISTER_PRESS_LSB (0xf8)
- #define BME280_REGISTER_PRESS_XLSB (0xf9)

- #define BME280_REGISTER_TEMP_MSB (0xfa)
- #define BME280_REGISTER_TEMP_LSB (0xfb)
- #define BME280_REGISTER_TEMP_XLSB (0xfc)
- #define BME280_REGISTER_HUM_MSB (0xfd)
- #define BME280_REGISTER_HUM_LSB (0xfe)
- #define BME280_REGISTER_CALIB00 (0x88)
- #define BME280_REGISTER_CALIB01 (0x89)
- #define BME280_REGISTER_CALIB02 (0x8a)
- #define BME280_REGISTER_CALIB03 (0x8b)
- #define BME280_REGISTER_CALIB04 (0x8c)
- #define BME280_REGISTER_CALIB05 (0x8d)
- #define BME280_REGISTER_CALIB06 (0x8e)
- #define BME280_REGISTER_CALIB07 (0x8f)
- #define BME280_REGISTER_CALIB08 (0x90)
- #define BME280_REGISTER_CALIB09 (0x91)
- #define BME280_REGISTER_CALIB10 (0x92)
- #define BME280_REGISTER_CALIB11 (0x93)
- #define BME280_REGISTER_CALIB12 (0x94)
- #define BME280_REGISTER_CALIB13 (0x95)
- #define BME280_REGISTER_CALIB14 (0x96)
- #define BME280_REGISTER_CALIB15 (0x97)
- #define BME280_REGISTER_CALIB16 (0x98)
- #define BME280_REGISTER_CALIB17 (0x99)
- #define BME280_REGISTER_CALIB18 (0x9a)
- #define BME280_REGISTER_CALIB19 (0x9b)
- #define BME280_REGISTER_CALIB20 (0x9c)
- #define BME280_REGISTER_CALIB21 (0x9d)
- #define BME280_REGISTER_CALIB22 (0x9e)
- #define BME280_REGISTER_CALIB23 (0x9f)
- #define BME280_REGISTER_CALIB24 (0xa0)
- #define BME280_REGISTER_CALIB25 (0xa1)
- #define BME280_REGISTER_CALIB26 (0xe1)
- #define BME280_REGISTER_CALIB27 (0xe2)
- #define BME280_REGISTER_CALIB28 (0xe3)
- #define BME280_REGISTER_CALIB29 (0xe4)
- #define BME280_REGISTER_CALIB30 (0xe5)
- #define BME280_REGISTER_CALIB31 (0xe6)
- #define BME280_REGISTER_CALIB32 (0xe7)
- #define BME280_REGISTER_CALIB33 (0xe8)
- #define BME280_REGISTER_CALIB34 (0xe9)
- #define BME280_REGISTER_CALIB35 (0xea)
- #define BME280_REGISTER_CALIB36 (0xeb)
- #define BME280_REGISTER_CALIB37 (0xec)
- #define BME280_REGISTER_CALIB38 (0xed)
- #define BME280_REGISTER_CALIB39 (0xee)
- #define BME280_REGISTER_CALIB40 (0xef)
- #define BME280_REGISTER_CALIB41 (0xf0)
- #define BME280_DATA_RESET (0xb6)
- #define BME280_DATA_ID (0x60)
- #define BME280_SETTINGS_OSRS_H_SKIPPED (0 << 0)
- #define BME280_SETTINGS_OSRS_H_1 (1 << 0)
- #define BME280_SETTINGS_OSRS_H_2 (2 << 0)
- #define BME280_SETTINGS_OSRS_H_4 (3 << 0)
- #define BME280_SETTINGS_OSRS_H_8 (4 << 0)
- #define BME280_SETTINGS_OSRS_H_16 (5 << 0)

- #define BME280_SETTINGS_OSRS_T_SKIPPED (0 << 5)
- #define BME280_SETTINGS_OSRS_T_1 (1 << 5)
- #define BME280_SETTINGS_OSRS_T_2 (2 << 5)
- #define BME280_SETTINGS_OSRS_T_4 (3 << 5)
- #define BME280_SETTINGS_OSRS_T_8 (4 << 5)
- #define BME280_SETTINGS_OSRS_T_16 (5 << 5)
- #define BME280_SETTINGS_OSRS_P_SKIPPED (0 << 2)
- #define BME280_SETTINGS_OSRS_P_1 (1 << 2)
- #define BME280_SETTINGS_OSRS_P_2 (2 << 2)
- #define BME280_SETTINGS_OSRS_P_4 (3 << 2)
- #define BME280_SETTINGS_OSRS_P_8 (4 << 2)
- #define BME280_SETTINGS_OSRS_P_16 (5 << 2)
- #define BME280_SETTINGS_MODE_SLEEP (0 << 0)
- #define BME280_SETTINGS_MODE_FORCE (1 << 0)
- #define BME280_SETTINGS_MODE_NORMAL (3 << 0)
- #define BME280_I2C_ACK_ENABLE (0x01)
- #define BME280_I2C_ACK_DISABLE (0x00)
- #define BME280_SIZE_HUM (0x02)
- #define BME280_SIZE_PRESS (0x03)
- #define BME280_SIZE_TEMP (0x03)
- #define BME280_SIZE_COMP (0x21)
- #define BME280_SETTINGS_DEFAULT

Enumerations

- enum bme280_result_t { BME280_RESULT_SUCCESS = 0 , BME280_RESULT_ERROR }

  Result codes for BME280 sensor operations.
- enum bme280_state_t { BME280_STATE_INIT = 0 , BME280_STATE_RESET , BME280_STATE_ID , BME280_STATE_FORCE_MODE , BME280_STATE_MEASURE_HUMIDITY , BME280_STATE_MEASURE_TEMPERATURE , BME280_STATE_MEASURE_PRE , BME280_STATE_GET_COMPENSATION_DATA , BME280_STATE_COMPENSATE_HUMIDITY , BME280_STATE_COMPENSATE_TEMPERATURE , BME280_STATE_COMPENSATE_PRESSURE , BME280_STATE_UNSET = 0xFF }

  State codes for BME280 sensor initialization and operations.

Functions

- bme280_result_t bme280_init (i2c_port_t i2c_num, const bme280_settings_t *settings)

  Initialize the BME280 sensor.
- bme280_result_t bme280_reset (i2c_port_t i2c_num)

  Reset the BME280 sensor.
- bme280_result_t bme280_id (i2c_port_t i2c_num, uint8_t *data, size_t data_len)

  Read the sensor ID.
- bme280_result_t bme280_force_mode (i2c_port_t i2c_num, const bme280_settings_t *settings)

  Set the sensor to force mode.
- bme280_result_t bme280_measure_humidity (i2c_port_t i2c_num, bme280_humidity_t *humidity)

  Measure humidity.
- bme280_result_t bme280_measure_temperature (i2c_port_t i2c_num, bme280_temperature_t *temperature)

  Measure temperature.
- bme280_result_t bme280_measure_pressure (i2c_port_t i2c_num, bme280_pressure_t *pressure)

  Measure pressure.

- bme280_result_t bme280_get_compensation_data (i2c_port_t i2c_num, bme280_compensator_t ∗compensator)

  Get sensor compensation data.
- bme280_result_t bme280_compensate_humidity (bme280_compensator_t ∗compensator, bme280_humidity_t ∗humidity)

  Compensate the humidity measurement.
- bme280_result_t bme280_compensate_temperature (bme280_compensator_t ∗compensator, bme280_temperature_t ∗temperature)

  Compensate the temperature measurement.
- bme280_result_t bme280_compensate_pressure (bme280_compensator_t ∗compensator, bme280_pressure_t ∗pressure)

  Compensate the pressure measurement.

4.1.1 Macro Definition Documentation

4.1.1.1 BME280_DATA_ID

```
#define BME280_DATA_ID (0x60)
```

4.1.1.2 BME280_DATA_RESET

```
#define BME280_DATA_RESET (0xb6)
```

4.1.1.3 BME280_I2C_ACK_DISABLE

```
#define BME280_I2C_ACK_DISABLE (0x00)
```

4.1.1.4 BME280_I2C_ACK_ENABLE

```
#define BME280_I2C_ACK_ENABLE (0x01)
```

4.1.1.5 BME280_I2C_ADDRESS

```
#define BME280_I2C_ADDRESS (0x76)
```

4.1.1.6 BME280_REGISTER_CALIB00

```
#define BME280_REGISTER_CALIB00 (0x88)
```

4.1.1.7 BME280_REGISTER_CALIB01

```
#define BME280_REGISTER_CALIB01 (0x89)
```

4.1.1.8 BME280_REGISTER_CALIB02

```
#define BME280_REGISTER_CALIB02 (0x8a)
```

4.1.1.9 BME280_REGISTER_CALIB03

```
#define BME280_REGISTER_CALIB03 (0x8b)
```

4.1.1.10 BME280_REGISTER_CALIB04

```
#define BME280_REGISTER_CALIB04 (0x8c)
```

4.1.1.11 BME280_REGISTER_CALIB05

```
#define BME280_REGISTER_CALIB05 (0x8d)
```

4.1.1.12 BME280_REGISTER_CALIB06

```
#define BME280_REGISTER_CALIB06 (0x8e)
```

4.1.1.13 BME280_REGISTER_CALIB07

```
#define BME280_REGISTER_CALIB07 (0x8f)
```

4.1.1.14 BME280_REGISTER_CALIB08

```
#define BME280_REGISTER_CALIB08 (0x90)
```

4.1.1.15 BME280_REGISTER_CALIB09

```
#define BME280_REGISTER_CALIB09 (0x91)
```

4.1.1.16 BME280_REGISTER_CALIB10

```
#define BME280_REGISTER_CALIB10 (0x92)
```

4.1.1.17 BME280_REGISTER_CALIB11

```
#define BME280_REGISTER_CALIB11 (0x93)
```

4.1.1.18 BME280_REGISTER_CALIB12

```
#define BME280_REGISTER_CALIB12 (0x94)
```

4.1.1.19 BME280_REGISTER_CALIB13

```
#define BME280_REGISTER_CALIB13 (0x95)
```

4.1.1.20 BME280_REGISTER_CALIB14

```
#define BME280_REGISTER_CALIB14 (0x96)
```

4.1.1.21 BME280_REGISTER_CALIB15

```
#define BME280_REGISTER_CALIB15 (0x97)
```

4.1.1.22 BME280_REGISTER_CALIB16

```
#define BME280_REGISTER_CALIB16 (0x98)
```

4.1.1.23 BME280_REGISTER_CALIB17

```
#define BME280_REGISTER_CALIB17 (0x99)
```

4.1.1.24 BME280_REGISTER_CALIB18

```
#define BME280_REGISTER_CALIB18 (0x9a)
```

4.1.1.25 BME280_REGISTER_CALIB19

```
#define BME280_REGISTER_CALIB19 (0x9b)
```

4.1.1.26 BME280_REGISTER_CALIB20

```
#define BME280_REGISTER_CALIB20 (0x9c)
```

4.1.1.27 BME280_REGISTER_CALIB21

```
#define BME280_REGISTER_CALIB21 (0x9d)
```

4.1.1.28 BME280_REGISTER_CALIB22

```
#define BME280_REGISTER_CALIB22 (0x9e)
```

4.1.1.29 BME280_REGISTER_CALIB23

```
#define BME280_REGISTER_CALIB23 (0x9f)
```

4.1.1.30 BME280_REGISTER_CALIB24

```
#define BME280_REGISTER_CALIB24 (0xa0)
```

4.1.1.31 BME280_REGISTER_CALIB25

```
#define BME280_REGISTER_CALIB25 (0xa1)
```

4.1.1.32 BME280_REGISTER_CALIB26

```
#define BME280_REGISTER_CALIB26 (0xe1)
```

4.1.1.33 BME280_REGISTER_CALIB27

```
#define BME280_REGISTER_CALIB27 (0xe2)
```

4.1.1.34 BME280_REGISTER_CALIB28

```
#define BME280_REGISTER_CALIB28 (0xe3)
```

4.1.1.35 BME280_REGISTER_CALIB29

```
#define BME280_REGISTER_CALIB29 (0xe4)
```

4.1.1.36 BME280_REGISTER_CALIB30

```
#define BME280_REGISTER_CALIB30 (0xe5)
```

4.1.1.37 BME280_REGISTER_CALIB31

```
#define BME280_REGISTER_CALIB31 (0xe6)
```

4.1.1.38 BME280_REGISTER_CALIB32

```
#define BME280_REGISTER_CALIB32 (0xe7)
```

4.1.1.39 BME280_REGISTER_CALIB33

```
#define BME280_REGISTER_CALIB33 (0xe8)
```

4.1.1.40 BME280_REGISTER_CALIB34

```
#define BME280_REGISTER_CALIB34 (0xe9)
```

4.1.1.41 BME280_REGISTER_CALIB35

```
#define BME280_REGISTER_CALIB35 (0xea)
```

4.1.1.42 BME280_REGISTER_CALIB36

```
#define BME280_REGISTER_CALIB36 (0xeb)
```

4.1.1.43 BME280_REGISTER_CALIB37

```
#define BME280_REGISTER_CALIB37 (0xec)
```

4.1.1.44 BME280_REGISTER_CALIB38

```
#define BME280_REGISTER_CALIB38 (0xed)
```

4.1.1.45 BME280_REGISTER_CALIB39

```
#define BME280_REGISTER_CALIB39 (0xee)
```

4.1.1.46 BME280_REGISTER_CALIB40

```
#define BME280_REGISTER_CALIB40 (0xef)
```

4.1.1.47 BME280_REGISTER_CALIB41

```
#define BME280_REGISTER_CALIB41 (0xf0)
```

4.1.1.48 BME280_REGISTER_CONFIG

```
#define BME280_REGISTER_CONFIG (0xf5)
```

4.1.1.49 BME280_REGISTER_CTRL_HUM

```
#define BME280_REGISTER_CTRL_HUM (0xf2)
```

4.1.1.50 BME280_REGISTER_CTRL_MEAS

```
#define BME280_REGISTER_CTRL_MEAS (0xf4)
```

4.1.1.51 BME280_REGISTER_HUM_LSB

```
#define BME280_REGISTER_HUM_LSB (0xfe)
```

4.1.1.52 BME280_REGISTER_HUM_MSB

```
#define BME280_REGISTER_HUM_MSB (0xfd)
```

4.1.1.53 BME280_REGISTER_ID

```
#define BME280_REGISTER_ID (0xd0)
```

4.1.1.54 BME280_REGISTER_PRESS_LSB

```
#define BME280_REGISTER_PRESS_LSB (0xf8)
```

4.1.1.55 BME280_REGISTER_PRESS_MSB

```
#define BME280_REGISTER_PRESS_MSB (0xf7)
```

4.1.1.56 BME280_REGISTER_PRESS_XLSB

```
#define BME280_REGISTER_PRESS_XLSB (0xf9)
```

4.1.1.57 BME280_REGISTER_RESET

```
#define BME280_REGISTER_RESET (0xe0)
```

4.1.1.58 BME280_REGISTER_STATUS

```
#define BME280_REGISTER_STATUS (0xf3)
```

4.1.1.59 BME280_REGISTER_TEMP_LSB

```
#define BME280_REGISTER_TEMP_LSB (0xfb)
```

4.1.1.60 BME280_REGISTER_TEMP_MSB

```
#define BME280_REGISTER_TEMP_MSB (0xfa)
```

4.1.1.61 BME280_REGISTER_TEMP_XLSB

```
#define BME280_REGISTER_TEMP_XLSB (0xfc)
```

4.1.1.62 BME280_SETTINGS_DEFAULT

```
#define BME280_SETTINGS_DEFAULT
```

Value:

```
  {                                                \
  .ctrl_hum = BME280_SETTINGS_OSRS_H_1,                                \
  .ctrl_meas = (BME280_SETTINGS_OSRS_T_1 | BME280_SETTINGS_OSRS_P_1 | \
                BME280_SETTINGS_MODE_FORCE),                           \
  .config = 0x00,                                                      \
}
```

Suggested settings for weather monitoring; Sensor mode: force mode, 1 sample per minute;

4.1.1.63 BME280_SETTINGS_MODE_FORCE

```
#define BME280_SETTINGS_MODE_FORCE (1 << 0)
```

4.1.1.64 BME280_SETTINGS_MODE_NORMAL

```
#define BME280_SETTINGS_MODE_NORMAL (3 << 0)
```

4.1.1.65 BME280_SETTINGS_MODE_SLEEP

```
#define BME280_SETTINGS_MODE_SLEEP (0 << 0)
```

4.1.1.66 BME280_SETTINGS_OSRS_H_1

```
#define BME280_SETTINGS_OSRS_H_1 (1 << 0)
```

4.1.1.67 BME280_SETTINGS_OSRS_H_16

```
#define BME280_SETTINGS_OSRS_H_16 (5 << 0)
```

4.1.1.68 BME280_SETTINGS_OSRS_H_2

```
#define BME280_SETTINGS_OSRS_H_2 (2 << 0)
```

4.1.1.69 BME280_SETTINGS_OSRS_H_4

```
#define BME280_SETTINGS_OSRS_H_4 (3 << 0)
```

4.1.1.70 BME280_SETTINGS_OSRS_H_8

```
#define BME280_SETTINGS_OSRS_H_8 (4 << 0)
```

4.1.1.71 BME280_SETTINGS_OSRS_H_SKIPPED

```
#define BME280_SETTINGS_OSRS_H_SKIPPED (0 << 0)
```

4.1.1.72 BME280_SETTINGS_OSRS_P_1

```
#define BME280_SETTINGS_OSRS_P_1 (1 << 2)
```

4.1.1.73 BME280_SETTINGS_OSRS_P_16

```
#define BME280_SETTINGS_OSRS_P_16 (5 << 2)
```

4.1.1.74 BME280_SETTINGS_OSRS_P_2

```
#define BME280_SETTINGS_OSRS_P_2 (2 << 2)
```

4.1.1.75 BME280_SETTINGS_OSRS_P_4

```
#define BME280_SETTINGS_OSRS_P_4 (3 << 2)
```

4.1.1.76 BME280_SETTINGS_OSRS_P_8

```
#define BME280_SETTINGS_OSRS_P_8 (4 << 2)
```

4.1.1.77 BME280_SETTINGS_OSRS_P_SKIPPED

```
#define BME280_SETTINGS_OSRS_P_SKIPPED (0 << 2)
```

4.1.1.78 BME280_SETTINGS_OSRS_T_1

```
#define BME280_SETTINGS_OSRS_T_1 (1 << 5)
```

4.1.1.79 BME280_SETTINGS_OSRS_T_16

```
#define BME280_SETTINGS_OSRS_T_16 (5 << 5)
```

4.1.1.80 BME280_SETTINGS_OSRS_T_2

```
#define BME280_SETTINGS_OSRS_T_2 (2 << 5)
```

4.1.1.81 BME280_SETTINGS_OSRS_T_4

```
#define BME280_SETTINGS_OSRS_T_4 (3 << 5)
```

4.1.1.82 BME280_SETTINGS_OSRS_T_8

```
#define BME280_SETTINGS_OSRS_T_8 (4 << 5)
```

4.1.1.83 BME280_SETTINGS_OSRS_T_SKIPPED

```
#define BME280_SETTINGS_OSRS_T_SKIPPED (0 << 5)
```

4.1.1.84 BME280_SIZE_COMP

```
#define BME280_SIZE_COMP (0x21)
```

4.1.1.85 BME280_SIZE_HUM

```
#define BME280_SIZE_HUM (0x02)
```

4.1.1.86 BME280_SIZE_PRESS

```
#define BME280_SIZE_PRESS (0x03)
```

4.1.1.87 BME280_SIZE_TEMP

```
#define BME280_SIZE_TEMP (0x03)
```

4.1.2 Enumeration Type Documentation

4.1.2.1 bme280_result_t

```
enum bme280_result_t
```

Result codes for BME280 sensor operations.

Enumerator

BME280_RESULT_SUCCESS	Operation was successful.
BME280_RESULT_ERROR	Operation encountered an error.

4.1.2.2 bme280_state_t

```
enum bme280_state_t
```

State codes for BME280 sensor initialization and operations.

Enumerator

BME280_STATE_INIT	Initial state.
BME280_STATE_RESET	Reset state.
BME280_STATE_ID	ID reading state.
BME280_STATE_FORCE_MODE	Force mode state.
BME280_STATE_MEASURE_HUMIDITY	Humidity measurement state.
BME280_STATE_MEASURE_TEMPERATURE	Temperature measurement state.
BME280_STATE_MEASURE_PRESSURE	Pressure measurement state.
BME280_STATE_GET_COMPENSATION_DATA	Compensation data retrieval state.
BME280_STATE_COMPENSATE_HUMIDITY	Humidity compensation state.
BME280_STATE_COMPENSATE_TEMPERATURE	Temperature compensation state.
BME280_STATE_COMPENSATE_PRESSURE	Pressure compensation state.
BME280_STATE_UNSET	Unset state.

4.1.3 Function Documentation

4.1.3.1 bme280_compensate_humidity()

```
bme280_result_t bme280_compensate_humidity (
            bme280_compensator_t * compensator,
            bme280_humidity_t * humidity)
```

Compensate the humidity measurement.

Parameters

in	*compensator*	Pointer to compensator structure with calibration data.
out	*humidity*	Pointer to humidity structure to store compensated value.

Returns

Result of the humidity compensation.

4.1.3.2 bme280_compensate_pressure()

```
bme280_result_t bme280_compensate_pressure (
            bme280_compensator_t * compensator,
            bme280_pressure_t * pressure)
```

Compensate the pressure measurement.

Parameters

in	*compensator*	Pointer to compensator structure with calibration data.
out	*pressure*	Pointer to pressure structure to store compensated value.

Returns

Result of the pressure compensation.

4.1.3.3 bme280_compensate_temperature()

```
bme280_result_t bme280_compensate_temperature (
            bme280_compensator_t * compensator,
            bme280_temperature_t * temperature)
```

Compensate the temperature measurement.

Parameters

in	*compensator*	Pointer to compensator structure with calibration data.
out	*temperature*	Pointer to temperature structure to store compensated value.

Returns

Result of the temperature compensation.

4.1.3.4 bme280_force_mode()

```
bme280_result_t bme280_force_mode (
            i2c_port_t i2c_num,
            const bme280_settings_t * settings)
```

Set the sensor to force mode.

Parameters

in	*i2c_num*	I2C port number.
in	*settings*	Pointer to sensor settings structure.

Returns

Result of the operation.

4.1.3.5 bme280_get_compensation_data()

```
bme280_result_t bme280_get_compensation_data (
            i2c_port_t i2c_num,
            bme280_compensator_t * compensator)
```

Get sensor compensation data.

Parameters

in	*i2c_num*	I2C port number.
out	*compensator*	Pointer to compensator structure to store the data.

Returns

Result of the compensation data retrieval.

4.1.3.6 bme280_id()

```
bme280_result_t bme280_id (
            i2c_port_t i2c_num,
            uint8_t * data,
            size_t data_len)
```

Read the sensor ID.

Parameters

in	*i2c_num*	I2C port number.
out	*data*	Pointer to buffer to store ID data.
in	*data_len*	Length of the buffer.

Returns

Result of the ID read operation.

4.1.3.7 bme280_init()

```
bme280_result_t bme280_init (
            i2c_port_t i2c_num,
            const bme280_settings_t * settings)
```

Initialize the BME280 sensor.

Parameters

in	*i2c_num*	I2C port number.
in	*settings*	Pointer to sensor settings structure.

Returns

Result of the initialization.

4.1.3.8 bme280_measure_humidity()

```
bme280_result_t bme280_measure_humidity (
            i2c_port_t i2c_num,
            bme280_humidity_t * humidity)
```

Measure humidity.

Parameters

in	*i2c_num*	I2C port number.
out	*humidity*	Pointer to humidity structure to store the measurement.

Returns

Result of the humidity measurement.

4.1.3.9 bme280_measure_pressure()

```
bme280_result_t bme280_measure_pressure (
            i2c_port_t i2c_num,
            bme280_pressure_t * pressure)
```

Measure pressure.

Parameters

in	*i2c_num*	I2C port number.
out	*pressure*	Pointer to pressure structure to store the measurement.

Returns

Result of the pressure measurement.

4.1.3.10 bme280_measure_temperature()

```
bme280_result_t bme280_measure_temperature (
            i2c_port_t i2c_num,
            bme280_temperature_t * temperature)
```

Measure temperature.

Parameters

in	*i2c_num*	I2C port number.
out	*temperature*	Pointer to temperature structure to store the measurement.

Returns

Result of the temperature measurement.

4.1.3.11 bme280_reset()

```
bme280_result_t bme280_reset (
            i2c_port_t i2c_num)
```

Reset the BME280 sensor.

Parameters

in	*i2c_num*	I2C port number.

Returns

Result of the reset operation.

4.2 bme280.h

Go to the documentation of this file.

```
00001 #ifndef INC_BME280_H
00002 #define INC_BME280_H
00003
00004 #include <stddef.h>
00005 #include <stdint.h>
00006 #include "hal/i2c_types.h"
00007 #include "i2c_controller.h"
00008
00009 #define BME280_I2C_ADDRESS  (0x76)
00010
00011 #define BME280_REGISTER_ID          (0xd0)
00012 #define BME280_REGISTER_RESET       (0xe0)
00013 #define BME280_REGISTER_CTRL_HUM    (0xf2)
00014 #define BME280_REGISTER_STATUS      (0xf3)
00015 #define BME280_REGISTER_CTRL_MEAS   (0xf4)
00016 #define BME280_REGISTER_CONFIG      (0xf5)
00017 #define BME280_REGISTER_PRESS_MSB   (0xf7)
00018 #define BME280_REGISTER_PRESS_LSB   (0xf8)
00019 #define BME280_REGISTER_PRESS_XLSB  (0xf9)
00020 #define BME280_REGISTER_TEMP_MSB    (0xfa)
00021 #define BME280_REGISTER_TEMP_LSB    (0xfb)
00022 #define BME280_REGISTER_TEMP_XLSB   (0xfc)
00023 #define BME280_REGISTER_HUM_MSB     (0xfd)
00024 #define BME280_REGISTER_HUM_LSB     (0xfe)
00025
00026 #define BME280_REGISTER_CALIB00   (0x88)
00027 #define BME280_REGISTER_CALIB01   (0x89)
00028 #define BME280_REGISTER_CALIB02   (0x8a)
00029 #define BME280_REGISTER_CALIB03   (0x8b)
00030 #define BME280_REGISTER_CALIB04   (0x8c)
00031 #define BME280_REGISTER_CALIB05   (0x8d)
00032 #define BME280_REGISTER_CALIB06   (0x8e)
00033 #define BME280_REGISTER_CALIB07   (0x8f)
00034 #define BME280_REGISTER_CALIB08   (0x90)
00035 #define BME280_REGISTER_CALIB09   (0x91)
00036 #define BME280_REGISTER_CALIB10     (0x92)
00037 #define BME280_REGISTER_CALIB11     (0x93)
00038 #define BME280_REGISTER_CALIB12     (0x94)
00039 #define BME280_REGISTER_CALIB13     (0x95)
00040 #define BME280_REGISTER_CALIB14     (0x96)
00041 #define BME280_REGISTER_CALIB15     (0x97)
00042 #define BME280_REGISTER_CALIB16     (0x98)
00043 #define BME280_REGISTER_CALIB17     (0x99)
00044 #define BME280_REGISTER_CALIB18     (0x9a)
00045 #define BME280_REGISTER_CALIB19     (0x9b)
00046 #define BME280_REGISTER_CALIB20     (0x9c)
00047 #define BME280_REGISTER_CALIB21     (0x9d)
00048 #define BME280_REGISTER_CALIB22     (0x9e)
00049 #define BME280_REGISTER_CALIB23     (0x9f)
00050 #define BME280_REGISTER_CALIB24     (0xa0)
00051 #define BME280_REGISTER_CALIB25     (0xa1)
00052 #define BME280_REGISTER_CALIB26     (0xe1)
00053 #define BME280_REGISTER_CALIB27     (0xe2)
00054 #define BME280_REGISTER_CALIB28     (0xe3)
00055 #define BME280_REGISTER_CALIB29     (0xe4)
00056 #define BME280_REGISTER_CALIB30     (0xe5)
00057 #define BME280_REGISTER_CALIB31     (0xe6)
00058 #define BME280_REGISTER_CALIB32     (0xe7)
00059 #define BME280_REGISTER_CALIB33     (0xe8)
00060 #define BME280_REGISTER_CALIB34     (0xe9)
00061 #define BME280_REGISTER_CALIB35     (0xea)
00062 #define BME280_REGISTER_CALIB36     (0xeb)
00063 #define BME280_REGISTER_CALIB37     (0xec)
00064 #define BME280_REGISTER_CALIB38     (0xed)
00065 #define BME280_REGISTER_CALIB39     (0xee)
```

```
00066 #define BME280_REGISTER_CALIB40     (0xef)
00067 #define BME280_REGISTER_CALIB41     (0xf0)
00068
00069 #define BME280_DATA_RESET (0xb6)
00070 #define BME280_DATA_ID    (0x60)
00071
00072 #define BME280_SETTINGS_OSRS_H_SKIPPED  (0 « 0)
00073 #define BME280_SETTINGS_OSRS_H_1        (1 « 0)
00074 #define BME280_SETTINGS_OSRS_H_2        (2 « 0)
00075 #define BME280_SETTINGS_OSRS_H_4        (3 « 0)
00076 #define BME280_SETTINGS_OSRS_H_8        (4 « 0)
00077 #define BME280_SETTINGS_OSRS_H_16       (5 « 0)
00078
00079 #define BME280_SETTINGS_OSRS_T_SKIPPED  (0 « 5)
00080 #define BME280_SETTINGS_OSRS_T_1        (1 « 5)
00081 #define BME280_SETTINGS_OSRS_T_2        (2 « 5)
00082 #define BME280_SETTINGS_OSRS_T_4        (3 « 5)
00083 #define BME280_SETTINGS_OSRS_T_8        (4 « 5)
00084 #define BME280_SETTINGS_OSRS_T_16       (5 « 5)
00085
00086 #define BME280_SETTINGS_OSRS_P_SKIPPED  (0 « 2)
00087 #define BME280_SETTINGS_OSRS_P_1        (1 « 2)
00088 #define BME280_SETTINGS_OSRS_P_2        (2 « 2)
00089 #define BME280_SETTINGS_OSRS_P_4        (3 « 2)
00090 #define BME280_SETTINGS_OSRS_P_8        (4 « 2)
00091 #define BME280_SETTINGS_OSRS_P_16       (5 « 2)
00092
00093 #define BME280_SETTINGS_MODE_SLEEP      (0 « 0)
00094 #define BME280_SETTINGS_MODE_FORCE      (1 « 0)
00095 #define BME280_SETTINGS_MODE_NORMAL     (3 « 0)
00096
00097 #define BME280_I2C_ACK_ENABLE   (0x01)
00098 #define BME280_I2C_ACK_DISABLE  (0x00)
00099
00100 #define BME280_SIZE_HUM   (0x02)
00101 #define BME280_SIZE_PRESS (0x03)
00102 #define BME280_SIZE_TEMP  (0x03)
00103 #define BME280_SIZE_COMP  (0x21)
00104
00108 typedef enum {
00109   BME280_RESULT_SUCCESS = 0,
00110   BME280_RESULT_ERROR,
00111 } bme280_result_t;
00112
00116 typedef enum {
00117   BME280_STATE_INIT = 0,
00118   BME280_STATE_RESET,
00119   BME280_STATE_ID,
00120   BME280_STATE_FORCE_MODE,
00121   BME280_STATE_MEASURE_HUMIDITY,
00122   BME280_STATE_MEASURE_TEMPERATURE,
00123   BME280_STATE_MEASURE_PRESSURE,
00124   BME280_STATE_GET_COMPENSATION_DATA,
00125   BME280_STATE_COMPENSATE_HUMIDITY,
00126   BME280_STATE_COMPENSATE_TEMPERATURE,
00127   BME280_STATE_COMPENSATE_PRESSURE,
00128   BME280_STATE_UNSET = 0xFF,
00129 } bme280_state_t;
00130
00134 typedef struct {
00135   uint8_t ctrl_hum;
00136   uint8_t ctrl_meas;
00137   uint8_t config;
00138 } bme280_settings_t;
00139
00143 typedef struct {
00144   uint8_t msb;
00145   uint8_t lsb;
00146   uint8_t xlsb;
00147   double  compensated;
00148 } bme280_pressure_t;
00149
00153 typedef struct {
00154   uint8_t msb;
00155   uint8_t lsb;
00156   uint8_t xlsb;
00157   double  compensated;
00158 } bme280_temperature_t;
00159
00163 typedef struct {
00164   uint8_t msb;
00165   uint8_t lsb;
00166   double  compensated;
00167 } bme280_humidity_t;
00168
00172 typedef struct {
00173   uint16_t  dig_t1;
```

```
00174   int16_t   dig_t2;
00175   int16_t   dig_t3;
00176   uint16_t  dig_p1;
00177   int16_t   dig_p2;
00178   int16_t   dig_p3;
00179   int16_t   dig_p4;
00180   int16_t   dig_p5;
00181   int16_t   dig_p6;
00182   int16_t   dig_p7;
00183   int16_t   dig_p8;
00184   int16_t   dig_p9;
00185   uint8_t   dig_h1;
00186   int16_t   dig_h2;
00187   uint8_t   dig_h3;
00188   int16_t   dig_h4;
00189   int16_t   dig_h5;
00190   int8_t    dig_h6;
00191 } bme280_compensator_t;
00192
00196 typedef struct {
00197   bme280_humidity_t humidity;
00198   bme280_pressure_t pressure;
00199   bme280_temperature_t temperature;
00200   bme280_compensator_t compensator;
00201 } bme280_measurements_t;
00202
00207 #define BME280_SETTINGS_DEFAULT {                                    \
00208   .ctrl_hum = BME280_SETTINGS_OSRS_H_1,                              \
00209   .ctrl_meas = (BME280_SETTINGS_OSRS_T_1 | BME280_SETTINGS_OSRS_P_1 | \
00210                 BME280_SETTINGS_MODE_FORCE),                         \
00211   .config = 0x00,                                                    \
00212 }
00213
00221 bme280_result_t bme280_init(i2c_port_t i2c_num, const bme280_settings_t *settings);
00222
00229 bme280_result_t bme280_reset(i2c_port_t i2c_num);
00230
00239 bme280_result_t bme280_id(i2c_port_t i2c_num, uint8_t *data, size_t data_len);
00240
00248 bme280_result_t bme280_force_mode(i2c_port_t i2c_num, const bme280_settings_t *settings);
00249
00257 bme280_result_t bme280_measure_humidity(i2c_port_t i2c_num, bme280_humidity_t *humidity);
00258
00266 bme280_result_t bme280_measure_temperature(i2c_port_t i2c_num,
00267                                            bme280_temperature_t *temperature);
00268
00276 bme280_result_t bme280_measure_pressure(i2c_port_t i2c_num, bme280_pressure_t *pressure);
00277
00285 bme280_result_t bme280_get_compensation_data(i2c_port_t i2c_num,
00286                                              bme280_compensator_t *compensator);
00287
00295 bme280_result_t bme280_compensate_humidity(bme280_compensator_t *compensator,
00296                                            bme280_humidity_t *humidity);
00297
00305 bme280_result_t bme280_compensate_temperature(bme280_compensator_t *compensator,
00306                                               bme280_temperature_t *temperature);
00307
00315 bme280_result_t bme280_compensate_pressure(bme280_compensator_t *compensator,
00316                                            bme280_pressure_t *pressure);
00317
00318 #endif // !INC_BME280_H
```

4.3 ether.h File Reference

```
#include "freertos/FreeRTOS.h"
#include "freertos/projdefs.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "esp_log.h"
#include "esp_task_wdt.h"
#include "driver/uart.h"
#include "hal/uart_types.h"
#include "portmacro.h"
#include "string.h"
#include "driver/gpio.h"
#include "pms7003.h"
```

```
#include "bme280.h"
#include "i2c_controller.h"
#include "mqtt_client.h"
#include "uart_controller.h"
#include "wifi_controller.h"
#include "mqtt_controller.h"
```

Data Structures

- struct ether_measurements_t
  Structure to store PMS7003 and BME280 sensors measurements.
- struct ether_descriptor_t
  Structure to hold various controller descriptors.
- struct ether_settings_t
  Structure for ETHER settings.
- struct ether_state_machine_t
  Structure to hold the state machine for PMS7003 and BME280 sensors.
- struct ether_t
  Main structure for ETHER containing measurements, descriptors, settings, and state machine.

Enumerations

- enum ether_result_t { ETHER_RESULT_SUCCESS = 0 , ETHER_RESULT_ERROR }
  Result codes for ETHER operations.

Functions

- ether_result_t ether_init (ether_t *ether)
  Initialize the ETHER system.

4.3.1 Enumeration Type Documentation

4.3.1.1 ether_result_t

```
enum ether_result_t
```

Result codes for ETHER operations.

Enumerator

ETHER_RESULT_SUCCESS	Operation was successful.
ETHER_RESULT_ERROR	Operation encountered an error.

4.3.2 Function Documentation

4.3.2.1 ether_init()

```
ether_result_t ether_init (
            ether_t * ether)
```

Initialize the ETHER system.

Parameters

out	*ether*	Pointer to the ETHER structure.

Returns

Result of the initialization.

4.4 ether.h

Go to the documentation of this file.

```
00001 #ifndef INC_ETHER_H
00002 #define INC_ETHER_H
00003
00004 #include "freertos/FreeRTOS.h"
00005 #include "freertos/projdefs.h"
00006 #include "freertos/task.h"
00007 #include "esp_system.h"
00008 #include "esp_log.h"
00009 #include "esp_task_wdt.h"
00010 #include "driver/uart.h"
00011 #include "hal/uart_types.h"
00012 #include "portmacro.h"
00013 #include "string.h"
00014 #include "driver/gpio.h"
00015 #include "pms7003.h"
00016 #include "bme280.h"
00017 #include "i2c_controller.h"
00018 #include "mqtt_client.h"
00019 #include "uart_controller.h"
00020 #include "wifi_controller.h"
00021 #include "mqtt_controller.h"
00022
00026 typedef enum {
00027   ETHER_RESULT_SUCCESS = 0,
00028   ETHER_RESULT_ERROR,
00029 } ether_result_t;
00030
00034 typedef struct {
00035   pms7003_measurements_t pms7003;
00036   bme280_measurements_t bme280;
00037 } ether_measurements_t;
00038
00042 typedef struct {
00043   i2c_controller_descriptor_t i2c_controller;
00044   mqtt_controller_descriptor_t mqtt_controller;
00045   uart_controller_descriptor_t uart_controller;
00046   wifi_controller_descriptor_t wifi_controller;
00047 } ether_descriptor_t;
00048
00052 typedef struct {
00053   bme280_settings_t bme280;
00054 } ether_settings_t;
00055
00059 typedef struct {
00060   pms7003_state_t pms7003;
00061   bme280_state_t bme280;
00062 } ether_state_machine_t;
00063
00067 typedef struct {
00068   ether_measurements_t measurements;
00069   ether_descriptor_t descriptor;
00070   ether_settings_t settings;
00071   ether_state_machine_t state_machine;
00072 } ether_t;
00073
00080 ether_result_t ether_init(ether_t *ether);
00081
00082 #endif // !INC_ETHER_H
```

4.5 i2c_controller.h File Reference

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

```
#include "driver/i2c.h"
#include "esp_log.h"
#include "hal/gpio_types.h"
#include "hal/i2c_types.h"
#include "soc/gpio_num.h"
#include "freertos/projdefs.h"
```

Data Structures

- struct i2c_controller_descriptor_t

  Structure for I2C controller descriptor.

Macros

- #define I2C_CONTROLLER_I2C_ACK_ENABLE (0x01)
- #define I2C_CONTROLLER_I2C_ACK_DISABLE (0x00)
- #define I2C_CONTROLLER_I2C_ACK (0x00)
- #define I2C_CONTROLLER_I2C_NACK (0x01)
- #define I2C_CONTROLLER_MASTER_SCL_IO (GPIO_NUM_22)
- #define I2C_CONTROLLER_MASTER_SDA_IO (GPIO_NUM_21)
- #define I2C_CONTROLLER_MASTER_FREQ_HZ (100000)
- #define I2C_CONTROLLER_MASTER_TX_BUF_DISABLE (0)
- #define I2C_CONTROLLER_MASTER_RX_BUF_DISABLE (0)
- #define I2C_CONTROLLER_CONFIG_DEFAULT

  Default configuration for the I2C controller.
- #define I2C_CONTROLLER_DESCRIPTOR_DEFAULT

  Default descriptor for the I2C controller.

Enumerations

- enum i2c_controller_result_t { I2C_CONTROLLER_RESULT_SUCCESS = 0 , I2C_CONTROLLER_RESULT_ERROR }

  Result codes for I2C controller operations.

Functions

- i2c_controller_result_t i2c_controller_init (const i2c_controller_descriptor_t *descriptor)

  Initialize the I2C controller.
- i2c_controller_result_t i2c_controller_send (i2c_port_t i2c_num, uint8_t address, uint8_t reg, const uint8_t *data, size_t data_len)

  Send data over I2C.
- i2c_controller_result_t i2c_controller_receive (i2c_port_t i2c_num, uint8_t address, uint8_t reg, uint8_t *data, size_t data_len)

  Receive data over I2C.

4.5.1 Macro Definition Documentation

4.5.1.1 I2C_CONTROLLER_CONFIG_DEFAULT

```
#define I2C_CONTROLLER_CONFIG_DEFAULT
```

Value:

```
  {                \
  .mode = I2C_MODE_MASTER,                            \
  .sda_io_num = I2C_CONTROLLER_MASTER_SDA_IO,         \
  .sda_pullup_en = GPIO_PULLUP_ENABLE,                \
  .scl_io_num = I2C_CONTROLLER_MASTER_SCL_IO,         \
  .scl_pullup_en = GPIO_PULLUP_ENABLE,                \
  .master.clk_speed = I2C_CONTROLLER_MASTER_FREQ_HZ,  \
}
```

Default configuration for the I2C controller.

4.5.1.2 I2C_CONTROLLER_DESCRIPTOR_DEFAULT

```
#define I2C_CONTROLLER_DESCRIPTOR_DEFAULT
```

Value:

```
  { \
  .config = I2C_CONTROLLER_CONFIG_DEFAULT,  \
  .i2c_num = I2C_NUM_0,                     \
}
```

Default descriptor for the I2C controller.

4.5.1.3 I2C_CONTROLLER_I2C_ACK

```
#define I2C_CONTROLLER_I2C_ACK (0x00)
```

4.5.1.4 I2C_CONTROLLER_I2C_ACK_DISABLE

```
#define I2C_CONTROLLER_I2C_ACK_DISABLE (0x00)
```

4.5.1.5 I2C_CONTROLLER_I2C_ACK_ENABLE

```
#define I2C_CONTROLLER_I2C_ACK_ENABLE (0x01)
```

4.5.1.6 I2C_CONTROLLER_I2C_NACK

```
#define I2C_CONTROLLER_I2C_NACK (0x01)
```

4.5.1.7 I2C_CONTROLLER_MASTER_FREQ_HZ

```
#define I2C_CONTROLLER_MASTER_FREQ_HZ (100000)
```

I2C master clock frequency.

4.5.1.8 I2C_CONTROLLER_MASTER_RX_BUF_DISABLE

```
#define I2C_CONTROLLER_MASTER_RX_BUF_DISABLE (0)
```

I2C master doesn't need buffer.

4.5.1.9 I2C_CONTROLLER_MASTER_SCL_IO

```
#define I2C_CONTROLLER_MASTER_SCL_IO (GPIO_NUM_22)
```

GPIO number for I2C master clock.

4.5.1.10 I2C_CONTROLLER_MASTER_SDA_IO

```
#define I2C_CONTROLLER_MASTER_SDA_IO (GPIO_NUM_21)
```

GPIO number for I2C master data.

4.5.1.11 I2C_CONTROLLER_MASTER_TX_BUF_DISABLE

```
#define I2C_CONTROLLER_MASTER_TX_BUF_DISABLE (0)
```

I2C master doesn't need buffer.

4.5.2 Enumeration Type Documentation

4.5.2.1 i2c_controller_result_t

```
enum i2c_controller_result_t
```

Result codes for I2C controller operations.

Enumerator

I2C_CONTROLLER_RESULT_SUCCESS	Operation was successful.
I2C_CONTROLLER_RESULT_ERROR	Operation encountered an error.

4.5.3 Function Documentation

4.5.3.1 i2c_controller_init()

```
i2c_controller_result_t i2c_controller_init (
            const i2c_controller_descriptor_t * descriptor)
```

Initialize the I2C controller.

Parameters

in	*descriptor*	Pointer to the I2C controller descriptor.

Returns

Result of the initialization.

4.5.3.2 i2c_controller_receive()

```
i2c_controller_result_t i2c_controller_receive (
            i2c_port_t i2c_num,
            uint8_t address,
            uint8_t reg,
            uint8_t * data,
            size_t data_len)
```

Receive data over I2C.

Parameters

in	*i2c_num*	I2C port number.
in	*address*	I2C address of the device.
in	*reg*	Register address to receive data from.
out	*data*	Pointer to the buffer to store received data.
in	*data_len*	Length of the data to receive.

Returns

Result of the receive operation.

4.5.3.3 i2c_controller_send()

```
i2c_controller_result_t i2c_controller_send (
            i2c_port_t i2c_num,
            uint8_t address,
            uint8_t reg,
            const uint8_t * data,
            size_t data_len)
```

Send data over I2C.

Parameters

in	*i2c_num*	I2C port number.
in	*address*	I2C address of the device.
in	*reg*	Register address to send data to.
in	*data*	Pointer to the data to send.
in	*data_len*	Length of the data to send.

Returns

Result of the send operation.

4.6 i2c_controller.h

Go to the documentation of this file.

```
00001 #ifndef INC_I2C_CONTROLLER_H
00002 #define INC_I2C_CONTROLLER_H
00003
00004 #include <stdint.h>
00005 #include <stddef.h>
00006 #include "driver/i2c.h"
00007 #include "esp_log.h"
00008 #include "hal/gpio_types.h"
00009 #include "hal/i2c_types.h"
00010 #include "soc/gpio_num.h"
00011 #include "freertos/projdefs.h"
00012
00013 #define I2C_CONTROLLER_I2C_ACK_ENABLE   (0x01)
00014 #define I2C_CONTROLLER_I2C_ACK_DISABLE  (0x00)
00015
00016 #define I2C_CONTROLLER_I2C_ACK   (0x00)
00017 #define I2C_CONTROLLER_I2C_NACK  (0x01)
00018
00019 #define I2C_CONTROLLER_MASTER_SCL_IO          (GPIO_NUM_22)
00020 #define I2C_CONTROLLER_MASTER_SDA_IO          (GPIO_NUM_21)
00021 #define I2C_CONTROLLER_MASTER_FREQ_HZ         (100000)
00022 #define I2C_CONTROLLER_MASTER_TX_BUF_DISABLE  (0)
00023 #define I2C_CONTROLLER_MASTER_RX_BUF_DISABLE  (0)
00028 typedef enum {
00029   I2C_CONTROLLER_RESULT_SUCCESS = 0,
00030   I2C_CONTROLLER_RESULT_ERROR,
00031 } i2c_controller_result_t;
00032
00036 typedef struct {
00037   i2c_config_t config;
00038   i2c_port_t i2c_num;
00039 } i2c_controller_descriptor_t;
00040
00044 #define I2C_CONTROLLER_CONFIG_DEFAULT {                \
00045   .mode = I2C_MODE_MASTER,                             \
00046   .sda_io_num = I2C_CONTROLLER_MASTER_SDA_IO,          \
00047   .sda_pullup_en = GPIO_PULLUP_ENABLE,                 \
00048   .scl_io_num = I2C_CONTROLLER_MASTER_SCL_IO,          \
00049   .scl_pullup_en = GPIO_PULLUP_ENABLE,                 \
00050   .master.clk_speed = I2C_CONTROLLER_MASTER_FREQ_HZ,   \
00051 }
00052
00056 #define I2C_CONTROLLER_DESCRIPTOR_DEFAULT { \
00057   .config = I2C_CONTROLLER_CONFIG_DEFAULT,  \
00058   .i2c_num = I2C_NUM_0,                     \
00059 }
00060
00067 i2c_controller_result_t i2c_controller_init(const i2c_controller_descriptor_t *descriptor);
00068
00079 i2c_controller_result_t i2c_controller_send(i2c_port_t i2c_num, uint8_t address,
00080                                             uint8_t reg, const uint8_t *data,
00081                                             size_t data_len);
00082
00093 i2c_controller_result_t i2c_controller_receive(i2c_port_t i2c_num, uint8_t address,
00094                                                uint8_t reg, uint8_t *data,
00095                                                size_t data_len);
00096
00097 #endif // !INC_I2C_CONTROLLER_H
```

4.7 mqtt_controller.h File Reference

```
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include "esp_wifi.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include "esp_event.h"
#include "esp_netif.h"
#include "lwip/sockets.h"
```

```
#include "lwip/dns.h"
#include "lwip/netdb.h"
#include "esp_log.h"
#include "mqtt_client.h"
```

Data Structures

- struct mqtt_controller_descriptor_t

  Structure for MQTT controller descriptor.

Macros

- #define MQTT_CONTROLLER_BROKER_ADDRESS_URI ("mqtt://192.168.235.80:1883")
- #define MQTT_CONTROLLER_MESSAGE_MAX_SIZE (256)
- #define MQTT_CONTROLLER_CONFIG_DEFAULT

  Default configuration for the MQTT controller.
- #define MQTT_CONTROLLER_DESCRIPTOR_DEFAULT

  Default descriptor for the MQTT controller.

Typedefs

- typedef void(∗ mqtt_controller_event_handler_t) (void ∗, esp_event_base_t, int32_t, void ∗)

  Function pointer type for MQTT event handler.

Enumerations

- enum mqtt_controller_result_t { MQTT_CONTROLLER_RESULT_SUCCESS = 0 , MQTT_CONTROLLER_RESULT_ERROR }

  Result codes for MQTT controller operations.

Functions

- void mqtt_controller_event_handler (void ∗handler_args, esp_event_base_t base, int32_t event_id, void ∗event_data)

  Handle MQTT events.
- mqtt_controller_result_t mqtt_controller_init (mqtt_controller_descriptor_t ∗mqtt_controller_descriptor)

  Initialize the MQTT controller.

4.7.1 Macro Definition Documentation

4.7.1.1 MQTT_CONTROLLER_BROKER_ADDRESS_URI

```
#define MQTT_CONTROLLER_BROKER_ADDRESS_URI ("mqtt://192.168.235.80:1883")
```

4.7.1.2 MQTT_CONTROLLER_CONFIG_DEFAULT

```
#define MQTT_CONTROLLER_CONFIG_DEFAULT
```

Value:

```
  {                   \
  .broker.address.uri = MQTT_CONTROLLER_BROKER_ADDRESS_URI, \
}
```

Default configuration for the MQTT controller.

4.7.1.3 MQTT_CONTROLLER_DESCRIPTOR_DEFAULT

```
#define MQTT_CONTROLLER_DESCRIPTOR_DEFAULT
```

Value:

```
  {       \
  .client_config = MQTT_CONTROLLER_CONFIG_DEFAULT,  \
  .event_handler = mqtt_controller_event_handler, \
}
```

Default descriptor for the MQTT controller.

4.7.1.4 MQTT_CONTROLLER_MESSAGE_MAX_SIZE

```
#define MQTT_CONTROLLER_MESSAGE_MAX_SIZE (256)
```

4.7.2 Typedef Documentation

4.7.2.1 mqtt_controller_event_handler_t

```
typedef void(* mqtt_controller_event_handler_t) (void *, esp_event_base_t, int32_t, void *)
```

Function pointer type for MQTT event handler.

Parameters

in	*handler_args*	Arguments passed to the handler.
in	*base*	Event base.
in	*event_id*	Event ID.
in	*event_data*	Event data.

4.7.3 Enumeration Type Documentation

4.7.3.1 mqtt_controller_result_t

```
enum mqtt_controller_result_t
```

Result codes for MQTT controller operations.

Enumerator

MQTT_CONTROLLER_RESULT_SUCCESS	Operation was successful.
MQTT_CONTROLLER_RESULT_ERROR	Operation encountered an error.

4.7.4 Function Documentation

4.7.4.1 mqtt_controller_event_handler()

```
void mqtt_controller_event_handler (
            void * handler_args,
            esp_event_base_t base,
            int32_t event_id,
            void * event_data)
```

Handle MQTT events.

Parameters

in	*handler_args*	Arguments passed to the handler.
in	*base*	Event base.
in	*event_id*	Event ID.
in	*event_data*	Event data.

4.7.4.2 mqtt_controller_init()

```
mqtt_controller_result_t mqtt_controller_init (
            mqtt_controller_descriptor_t * mqtt_controller_descriptor)
```

Initialize the MQTT controller.

Parameters

out	*mqtt_controller_descriptor*	Pointer to the MQTT controller descriptor.

Returns

Result of the initialization.

4.8 mqtt_controller.h

Go to the documentation of this file.

```
00001 #ifndef INC_MQTT_CONTROLLER_H
00002 #define INC_MQTT_CONTROLLER_H
00003
00004 #include <stdio.h>
00005 #include <stdint.h>
00006 #include <stddef.h>
00007 #include <string.h>
00008 #include "esp_wifi.h"
00009 #include "esp_system.h"
00010 #include "nvs_flash.h"
00011 #include "esp_event.h"
00012 #include "esp_netif.h"
00013
00014 #include "lwip/sockets.h"
00015 #include "lwip/dns.h"
00016 #include "lwip/netdb.h"
00017
00018 #include "esp_log.h"
00019 #include "mqtt_client.h"
00020
00021 #define MQTT_CONTROLLER_BROKER_ADDRESS_URI   ("mqtt://192.168.235.80:1883")
00022 #define MQTT_CONTROLLER_MESSAGE_MAX_SIZE     (256)
00023
00032 typedef void (*mqtt_controller_event_handler_t)(void *, esp_event_base_t, int32_t, void *);
00033
00037 typedef enum {
00038   MQTT_CONTROLLER_RESULT_SUCCESS = 0,
00039   MQTT_CONTROLLER_RESULT_ERROR,
00040 } mqtt_controller_result_t;
00041
00045 typedef struct {
00046   esp_mqtt_client_config_t client_config;
00047   esp_mqtt_client_handle_t client_handle;
00048   mqtt_controller_event_handler_t event_handler;
00049 } mqtt_controller_descriptor_t;
00050
00054 #define MQTT_CONTROLLER_CONFIG_DEFAULT  {                   \
00055   .broker.address.uri = MQTT_CONTROLLER_BROKER_ADDRESS_URI, \
00056 }
00057
00061 #define MQTT_CONTROLLER_DESCRIPTOR_DEFAULT  {       \
00062   .client_config = MQTT_CONTROLLER_CONFIG_DEFAULT,  \
00063   .event_handler = mqtt_controller_event_handler, \
00064 }
00065
00074 void mqtt_controller_event_handler(void *handler_args, esp_event_base_t base,
00075                                    int32_t event_id, void *event_data);
00076
00083 mqtt_controller_result_t mqtt_controller_init(mqtt_controller_descriptor_t
      *mqtt_controller_descriptor);
00084
00085 #endif // !INC_MQTT_CONTROLLER_H
```

4.9 pms7003.h File Reference

```
#include <stdint.h>
#include "driver/uart.h"
```

Data Structures

- union pms7003_frame_request_t

  Union representing a PMS7003 frame request.
- union pms7003_frame_answer_t

  Union representing a PMS7003 frame answer.
- struct pms7003_measurements_t

  Structure representing PMS7003 measurements.

Macros

- #define PMS7003_START_CHARACTER_1 (0x42)
- #define PMS7003_START_CHARACTER_2 (0x4d)
- #define PMS7003_CMD_READ (0xe2)
- #define PMS7003_CMD_CHANGE_MODE (0xe1)
- #define PMS7003_CMD_SLEEP_SET (0xe4)
- #define PMS7003_FRAME_REQUEST_SIZE (0x07)
- #define PMS7003_FRAME_ANSWER_SIZE (0x20)
- #define PMS7003_FRAME_CHECK_CODE_SIZE (0x1e)
- #define PMS7003_FRAME_BYTE_SIZE (0x01)
- #define PMS7003_UART_WAIT_TIMEOUT_MS (0x64)
- #define PMS7003_FRAME_READ

  Default frame for reading PMS7003 sensor data.
- #define PMS7003_FRAME_CHANGE_MODE_PASSIVE

  Default frame for changing PMS7003 sensor mode to passive.
- #define PMS7003_FRAME_CHANGE_MODE_ACTIVE

  Default frame for changing PMS7003 sensor mode to active.
- #define PMS7003_FRAME_SLEEP

  Default frame for putting PMS7003 sensor to sleep.
- #define PMS7003_FRAME_WAKEUP

  Default frame for waking PMS7003 sensor up.

Typedefs

- typedef int32_t(∗ pms7003_callback_sent_t) (uart_port_t)

  Callback function type for handling sent frames.
- typedef int32_t(∗ pms7003_callback_received_t) (uart_port_t, pms7003_frame_answer_t ∗)

  Callback function type for handling received frames.

Enumerations

- enum pms7003_result_t {
  PMS7003_RESULT_SUCCESS = 0 , PMS7003_RESULT_ERROR , PMS7003_RESULT_PARTIAL_SENT ,
  PMS7003_RESULT_PARTIAL_RECEIVED ,
  PMS7003_RESULT_WRONG_CHECK_CODE }

  Result codes for PMS7003 sensor operations.
- enum pms7003_status_t { PMS7003_STATUS_OK = 0 , PMS7003_STATUS_READY }

  Status codes for PMS7003 sensor.
- enum pms7003_state_t {
  PMS7003_STATE_READ_REQUEST = 0 , PMS7003_STATE_CHANGE_MODE_PASSIVE , PMS7003_STATE_CHANGE_MO
  , PMS7003_STATE_SLEEP ,
  PMS7003_STATE_WAKEUP , PMS7003_STATE_READ , PMS7003_STATE_UNSET = 0xFF }

  States of the PMS7003 sensor.

Functions

- pms7003_result_t pms7003_frame_send (const pms7003_callback_sent_t handler, uart_port_t uart_num)

  Send a PMS7003 frame.
- pms7003_result_t pms7003_frame_receive (const pms7003_callback_received_t handler, uart_port_t uart_num, pms7003_frame_answer_t *frame)

  Receive a PMS7003 frame.
- int32_t pms7003_read_request (uart_port_t uart_num)

  Send a read request to the PMS7003 sensor.
- int32_t pms7003_change_mode_passive (uart_port_t uart_num)

  Change the PMS7003 mode to passive.
- int32_t pms7003_change_mode_active (uart_port_t uart_num)

  Change the PMS7003 mode to active.
- int32_t pms7003_sleep (uart_port_t uart_num)

  Put the PMS7003 sensor to sleep.
- int32_t pms7003_wakeup (uart_port_t uart_num)

  Wake the PMS7003 sensor up.
- int32_t pms7003_read (uart_port_t uart_num, pms7003_frame_answer_t *frame)

  Read data from the PMS7003 sensor.

Variables

- const uint8_t start_byte_1
- const uint8_t start_byte_2
- uint8_t command
- uint8_t data_h
- uint8_t data_l
- uint8_t lrch
- uint8_t lrcl
- uint16_t length
- uint16_t data_pm1_standard
- uint16_t data_pm25_standard
- uint16_t data_pm10_standard
- uint16_t data_pm1_atmospheric
- uint16_t data_pm25_atmospheric
- uint16_t data_concentration_unit
- uint16_t data_particles_300nm
- uint16_t data_particles_500nm
- uint16_t data_particles_1000nm
- uint16_t data_particles_2500nm
- uint16_t data_particles_5000nm
- uint16_t data_particles_10000nm
- uint16_t reserved
- uint16_t check_code

4.9.1 Macro Definition Documentation

4.9.1.1 PMS7003_CMD_CHANGE_MODE

```
#define PMS7003_CMD_CHANGE_MODE (0xe1)
```

4.9.1.2 PMS7003_CMD_READ

```
#define PMS7003_CMD_READ (0xe2)
```

4.9.1.3 PMS7003_CMD_SLEEP_SET

```
#define PMS7003_CMD_SLEEP_SET (0xe4)
```

4.9.1.4 PMS7003_FRAME_ANSWER_SIZE

```
#define PMS7003_FRAME_ANSWER_SIZE (0x20)
```

4.9.1.5 PMS7003_FRAME_BYTE_SIZE

```
#define PMS7003_FRAME_BYTE_SIZE (0x01)
```

4.9.1.6 PMS7003_FRAME_CHANGE_MODE_ACTIVE

```
#define PMS7003_FRAME_CHANGE_MODE_ACTIVE
```

Value:

```
  {    \
  .start_byte_1 = PMS7003_START_CHARACTER_1,  \
  .start_byte_2 = PMS7003_START_CHARACTER_2,  \
  .command = PMS7003_CMD_CHANGE_MODE,         \
  .data_h = 0x00,                             \
  .data_l = 0x01,                             \
  .lrch = 0x01,                               \
  .lrcl = 0x71,                               \
}
```

Default frame for changing PMS7003 sensor mode to active.

4.9.1.7 PMS7003_FRAME_CHANGE_MODE_PASSIVE

```
#define PMS7003_FRAME_CHANGE_MODE_PASSIVE
```

Value:

```
  {   \
  .start_byte_1 = PMS7003_START_CHARACTER_1,  \
  .start_byte_2 = PMS7003_START_CHARACTER_2,  \
  .command = PMS7003_CMD_CHANGE_MODE,         \
  .data_h = 0x00,                             \
  .data_l = 0x00,                             \
  .lrch = 0x01,                               \
  .lrcl = 0x70,                               \
}
```

Default frame for changing PMS7003 sensor mode to passive.

4.9.1.8 PMS7003_FRAME_CHECK_CODE_SIZE

```
#define PMS7003_FRAME_CHECK_CODE_SIZE (0x1e)
```

4.9.1.9 PMS7003_FRAME_READ

```
#define PMS7003_FRAME_READ
```

Value:

```
  {                 \
  .start_byte_1 = PMS7003_START_CHARACTER_1,  \
  .start_byte_2 = PMS7003_START_CHARACTER_2,  \
  .command = PMS7003_CMD_READ,                \
  .data_h = 0x00,                             \
  .data_l = 0x00,                             \
  .lrch = 0x01,                               \
  .lrcl = 0x71,                               \
}
```

Default frame for reading PMS7003 sensor data.

4.9.1.10 PMS7003_FRAME_REQUEST_SIZE

```
#define PMS7003_FRAME_REQUEST_SIZE (0x07)
```

4.9.1.11 PMS7003_FRAME_SLEEP

```
#define PMS7003_FRAME_SLEEP
```

Value:

```
  {                 \
  .start_byte_1 = PMS7003_START_CHARACTER_1,  \
  .start_byte_2 = PMS7003_START_CHARACTER_2,  \
  .command = PMS7003_CMD_SLEEP_SET,           \
  .data_h = 0x00,                             \
  .data_l = 0x00,                             \
  .lrch = 0x01,                               \
  .lrcl = 0x73,                               \
}
```

Default frame for putting PMS7003 sensor to sleep.

4.9.1.12 PMS7003_FRAME_WAKEUP

```
#define PMS7003_FRAME_WAKEUP
```

Value:

```
  {                 \
  .start_byte_1 = PMS7003_START_CHARACTER_1,  \
  .start_byte_2 = PMS7003_START_CHARACTER_2,  \
  .command = PMS7003_CMD_SLEEP_SET,           \
  .data_h = 0x00,                             \
  .data_l = 0x01,                             \
  .lrch = 0x01,                               \
  .lrcl = 0x74,                               \
}
```

Default frame for waking PMS7003 sensor up.

4.9.1.13 PMS7003_START_CHARACTER_1

```
#define PMS7003_START_CHARACTER_1 (0x42)
```

4.9.1.14 PMS7003_START_CHARACTER_2

```
#define PMS7003_START_CHARACTER_2 (0x4d)
```

4.9.1.15 PMS7003_UART_WAIT_TIMEOUT_MS

```
#define PMS7003_UART_WAIT_TIMEOUT_MS (0x64)
```

4.9.2 Typedef Documentation

4.9.2.1 pms7003_callback_received_t

```
typedef int32_t(* pms7003_callback_received_t) (uart_port_t, pms7003_frame_answer_t *)
```

Callback function type for handling received frames.

Parameters

in	*uart_num*	UART port number.
out	*frame*	Pointer to the frame answer structure.

Returns

Status code of the operation.

4.9.2.2 pms7003_callback_sent_t

```
typedef int32_t(* pms7003_callback_sent_t) (uart_port_t)
```

Callback function type for handling sent frames.

Parameters

in	*uart_num*	UART port number.

Returns

Status code of the operation.

4.9.3 Enumeration Type Documentation

4.9.3.1 pms7003_result_t

```
enum pms7003_result_t
```

Result codes for PMS7003 sensor operations.

Enumerator

PMS7003_RESULT_SUCCESS	Operation was successful.
PMS7003_RESULT_ERROR	Operation encountered an error.
PMS7003_RESULT_PARTIAL_SENT	Partial data sent.
PMS7003_RESULT_PARTIAL_RECEIVED	Partial data received.
PMS7003_RESULT_WRONG_CHECK_CODE	Wrong checksum.

4.9.3.2 pms7003_state_t

```
enum pms7003_state_t
```

States of the PMS7003 sensor.

Enumerator

PMS7003_STATE_READ_REQUEST	Read request state.
PMS7003_STATE_CHANGE_MODE_PASSIVE	Change to passive mode state.
PMS7003_STATE_CHANGE_MODE_ACTIVE	Change to active mode state.
PMS7003_STATE_SLEEP	Sleep state.
PMS7003_STATE_WAKEUP	Wakeup state.
PMS7003_STATE_READ	Read state.
PMS7003_STATE_UNSET	Unset state.

4.9.3.3 pms7003_status_t

```
enum pms7003_status_t
```

Status codes for PMS7003 sensor.

Enumerator

PMS7003_STATUS_OK	Status OK.
PMS7003_STATUS_READY	Status Ready.

4.9.4 Function Documentation

4.9.4.1 pms7003_change_mode_active()

```
int32_t pms7003_change_mode_active (
            uart_port_t uart_num)
```

Change the PMS7003 mode to active.

Parameters

in	*uart_num*	UART port number.

Returns

Status code of the operation.

4.9.4.2 pms7003_change_mode_passive()

```
int32_t pms7003_change_mode_passive (
            uart_port_t uart_num)
```

Change the PMS7003 mode to passive.

Parameters

in	*uart_num*	UART port number.

Returns

Status code of the operation.

4.9.4.3 pms7003_frame_receive()

```
pms7003_result_t pms7003_frame_receive (
            const pms7003_callback_received_t handler,
            uart_port_t uart_num,
            pms7003_frame_answer_t * frame)
```

Receive a PMS7003 frame.

Parameters

in	*handler*	Callback function to handle the received frame.
in	*uart_num*	UART port number.
out	*frame*	Pointer to the frame answer structure.

Returns

Result of the receive operation.

4.9.4.4 pms7003_frame_send()

```
pms7003_result_t pms7003_frame_send (
            const pms7003_callback_sent_t handler,
            uart_port_t uart_num)
```

Send a PMS7003 frame.

Parameters

in	*handler*	Callback function to handle the sent frame.
in	*uart_num*	UART port number.

Returns

Result of the send operation.

4.9.4.5 pms7003_read()

```
int32_t pms7003_read (
            uart_port_t uart_num,
            pms7003_frame_answer_t * frame)
```

Read data from the PMS7003 sensor.

Parameters

in	*uart_num*	UART port number.
out	*frame*	Pointer to the frame answer structure.

Returns

Status code of the operation.

4.9.4.6 pms7003_read_request()

```
int32_t pms7003_read_request (
            uart_port_t uart_num)
```

Send a read request to the PMS7003 sensor.

Parameters

in	*uart_num*	UART port number.

Returns

Status code of the operation.

4.9.4.7 pms7003_sleep()

```
int32_t pms7003_sleep (
            uart_port_t uart_num)
```

Put the PMS7003 sensor to sleep.

Parameters

in	*uart_num*	UART port number.

Returns

Status code of the operation.

4.9.4.8 pms7003_wakeup()

```
int32_t pms7003_wakeup (
            uart_port_t uart_num)
```

Wake the PMS7003 sensor up.

Parameters

in	*uart_num*	UART port number.

Returns

Status code of the operation.

4.9.5 Variable Documentation

4.9.5.1 check_code

```
uint16_t check_code
```

Checksum code.

4.9.5.2 command

```
uint8_t command
```

Command byte.

4.9.5.3 data_concentration_unit

```
uint16_t data_concentration_unit
```

Concentration unit.

4.9.5.4 data_h

```
uint8_t data_h
```

Data high byte.

4.9.5.5 data_l

```
uint8_t data_l
```

Data low byte.

4.9.5.6 data_particles_10000nm

```
uint16_t data_particles_10000nm
```

Particles count for 10µm.

4.9.5.7 data_particles_1000nm

```
uint16_t data_particles_1000nm
```

Particles count for 1.0µm.

4.9.5.8 data_particles_2500nm

```
uint16_t data_particles_2500nm
```

Particles count for 2.5µm.

4.9.5.9 data_particles_300nm

```
uint16_t data_particles_300nm
```

Particles count for 0.3µm.

4.9.5.10 data_particles_5000nm

```
uint16_t data_particles_5000nm
```

Particles count for 5.0µm.

4.9.5.11 data_particles_500nm

```
uint16_t data_particles_500nm
```

Particles count for 0.5µm.

4.9.5.12 data_pm10_standard

```
uint16_t data_pm10_standard
```

PM10 concentration (standard particles).

4.9.5.13 data_pm1_atmospheric

```
uint16_t data_pm1_atmospheric
```

PM1.0 concentration (atmospheric particles).

4.9.5.14 data_pm1_standard

```
uint16_t data_pm1_standard
```

PM1.0 concentration (standard particles).

4.9.5.15 data_pm25_atmospheric

```
uint16_t data_pm25_atmospheric
```

PM2.5 concentration (atmospheric particles).

4.9.5.16 data_pm25_standard

```
uint16_t data_pm25_standard
```

PM2.5 concentration (standard particles).

4.9.5.17 length

```
uint16_t length
```

Length of the frame.

4.9.5.18 lrch

```
uint8_t lrch
```

High byte of the checksum.

4.9.5.19 lrcl

```
uint8_t lrcl
```

Low byte of the checksum.

4.9.5.20 reserved

```
uint16_t reserved
```

Reserved bytes.

4.9.5.21 start_byte_1

```
uint8_t start_byte_1
```

Start byte 1.

4.9.5.22 start_byte_2

```
uint8_t start_byte_2
```

Start byte 2.

4.10 pms7003.h

Go to the documentation of this file.

```
00001 #ifndef INC_PMS7003_H
00002 #define INC_PMS7003_H
00003
00004 #include <stdint.h>
00005 #include "driver/uart.h"
00006
00007 #define PMS7003_START_CHARACTER_1 (0x42)
00008 #define PMS7003_START_CHARACTER_2 (0x4d)
00009 #define PMS7003_CMD_READ          (0xe2)
00010 #define PMS7003_CMD_CHANGE_MODE   (0xe1)
00011 #define PMS7003_CMD_SLEEP_SET     (0xe4)
00012
00013 #define PMS7003_FRAME_REQUEST_SIZE    (0x07)
00014 #define PMS7003_FRAME_ANSWER_SIZE     (0x20)
00015 #define PMS7003_FRAME_CHECK_CODE_SIZE (0x1e)
00016 #define PMS7003_FRAME_BYTE_SIZE       (0x01)
00017
00018 #define PMS7003_UART_WAIT_TIMEOUT_MS  (0x64)
00019
00023 typedef union {
00024   struct {
00025     const uint8_t start_byte_1;
00026     const uint8_t start_byte_2;
00027     uint8_t command;
00028     uint8_t data_h;
00029     uint8_t data_l;
00030     uint8_t lrch;
00031     uint8_t lrcl;
00032     /* Verify bytes: add all the bytes except verify bytes. */
00033   } __attribute__((packed));  /* Ensure no padding between members. */
00034   uint8_t buffer_request[PMS7003_FRAME_REQUEST_SIZE];
00035 } pms7003_frame_request_t;
00036
00040 typedef union {
00041   struct {
00042     uint8_t start_byte_1;
00043     uint8_t start_byte_2;
00044     uint16_t length;
00045     uint16_t data_pm1_standard;
00046     uint16_t data_pm25_standard;
00047     uint16_t data_pm10_standard;
00048     uint16_t data_pm1_atmospheric;
00049     uint16_t data_pm25_atmospheric;
00050     uint16_t data_concentration_unit;
00051     uint16_t data_particles_300nm;
00052     uint16_t data_particles_500nm;
00053     uint16_t data_particles_1000nm;
00054     uint16_t data_particles_2500nm;
00055     uint16_t data_particles_5000nm;
00056     uint16_t data_particles_10000nm;
00057     uint16_t reserved;
00058     uint16_t check_code;
00059   } __attribute__((packed));  /* Ensure no padding between members. */
00060   uint8_t buffer_answer[PMS7003_FRAME_ANSWER_SIZE];
00061 } pms7003_frame_answer_t;
00062
00066 typedef struct {
00067   uint16_t pm1;
```

```
00068   uint16_t pm25;
00069   uint16_t pm10;
00070 } pms7003_measurements_t;
00071
00075 typedef enum {
00076   PMS7003_RESULT_SUCCESS = 0,
00077   PMS7003_RESULT_ERROR,
00078   PMS7003_RESULT_PARTIAL_SENT,
00079   PMS7003_RESULT_PARTIAL_RECEIVED,
00080   PMS7003_RESULT_WRONG_CHECK_CODE,
00081 } pms7003_result_t;
00082
00086 typedef enum {
00087   PMS7003_STATUS_OK = 0,
00088   PMS7003_STATUS_READY,
00089 } pms7003_status_t;
00090
00094 typedef enum {
00095   PMS7003_STATE_READ_REQUEST = 0,
00096   PMS7003_STATE_CHANGE_MODE_PASSIVE,
00097   PMS7003_STATE_CHANGE_MODE_ACTIVE,
00098   PMS7003_STATE_SLEEP,
00099   PMS7003_STATE_WAKEUP,
00100   PMS7003_STATE_READ,
00101   PMS7003_STATE_UNSET = 0xFF,
00102 } pms7003_state_t;
00103
00107 #define PMS7003_FRAME_READ {                  \
00108   .start_byte_1 = PMS7003_START_CHARACTER_1,  \
00109   .start_byte_2 = PMS7003_START_CHARACTER_2,  \
00110   .command = PMS7003_CMD_READ,                \
00111   .data_h = 0x00,                             \
00112   .data_l = 0x00,                             \
00113   .lrch = 0x01,                               \
00114   .lrcl = 0x71,                               \
00115 }
00116
00120 #define PMS7003_FRAME_CHANGE_MODE_PASSIVE {   \
00121   .start_byte_1 = PMS7003_START_CHARACTER_1,  \
00122   .start_byte_2 = PMS7003_START_CHARACTER_2,  \
00123   .command = PMS7003_CMD_CHANGE_MODE,         \
00124   .data_h = 0x00,                             \
00125   .data_l = 0x00,                             \
00126   .lrch = 0x01,                               \
00127   .lrcl = 0x70,                               \
00128 }
00129
00133 #define PMS7003_FRAME_CHANGE_MODE_ACTIVE {    \
00134   .start_byte_1 = PMS7003_START_CHARACTER_1,  \
00135   .start_byte_2 = PMS7003_START_CHARACTER_2,  \
00136   .command = PMS7003_CMD_CHANGE_MODE,         \
00137   .data_h = 0x00,                             \
00138   .data_l = 0x01,                             \
00139   .lrch = 0x01,                               \
00140   .lrcl = 0x71,                               \
00141 }
00142
00146 #define PMS7003_FRAME_SLEEP {                 \
00147   .start_byte_1 = PMS7003_START_CHARACTER_1,  \
00148   .start_byte_2 = PMS7003_START_CHARACTER_2,  \
00149   .command = PMS7003_CMD_SLEEP_SET,           \
00150   .data_h = 0x00,                             \
00151   .data_l = 0x00,                             \
00152   .lrch = 0x01,                               \
00153   .lrcl = 0x73,                               \
00154 }
00155
00159 #define PMS7003_FRAME_WAKEUP {                \
00160   .start_byte_1 = PMS7003_START_CHARACTER_1,  \
00161   .start_byte_2 = PMS7003_START_CHARACTER_2,  \
00162   .command = PMS7003_CMD_SLEEP_SET,           \
00163   .data_h = 0x00,                             \
00164   .data_l = 0x01,                             \
00165   .lrch = 0x01,                               \
00166   .lrcl = 0x74,                               \
00167 }
00168
00175 typedef int32_t (*pms7003_callback_sent_t)(uart_port_t);
00176
00184 typedef int32_t (*pms7003_callback_received_t)(uart_port_t, pms7003_frame_answer_t *);
00185
00193 pms7003_result_t pms7003_frame_send(const pms7003_callback_sent_t handler, uart_port_t uart_num);
00194
00203 pms7003_result_t pms7003_frame_receive(const pms7003_callback_received_t handler,
00204                                        uart_port_t uart_num, pms7003_frame_answer_t *frame);
00205
00212 int32_t pms7003_read_request(uart_port_t uart_num);
```

```
00213
00220 int32_t pms7003_change_mode_passive(uart_port_t uart_num);
00221
00228 int32_t pms7003_change_mode_active(uart_port_t uart_num);
00229
00236 int32_t pms7003_sleep(uart_port_t uart_num);
00237
00244 int32_t pms7003_wakeup(uart_port_t uart_num);
00245
00253 int32_t pms7003_read(uart_port_t uart_num, pms7003_frame_answer_t *frame);
00254
00255 #endif // !INC_PMS7003_H
```

4.11 state_machine.h File Reference

```
#include "ether.h"
```

Functions

- bme280_result_t state_machine_bme280_init (ether_t *ether)

  Initialize the BME280 sensor within the state machine.
- bme280_result_t state_machine_bme280_reset (ether_t *ether)

  Reset the BME280 sensor within the state machine.
- bme280_result_t state_machine_bme280_id (ether_t *ether)

  Read the ID of the BME280 sensor within the state machine.
- bme280_result_t state_machine_bme280_force_mode (ether_t *ether)

  Set the BME280 sensor to force mode within the state machine.
- bme280_result_t state_machine_bme280_measure_humidity (ether_t *ether)

  Measure humidity with the BME280 sensor within the state machine.
- bme280_result_t state_machine_bme280_measure_temperature (ether_t *ether)

  Measure temperature with the BME280 sensor within the state machine.
- bme280_result_t state_machine_bme280_measure_pressure (ether_t *ether)

  Measure pressure with the BME280 sensor within the state machine.
- bme280_result_t state_machine_bme280_get_compensation_data (ether_t *ether)

  Get compensation data from the BME280 sensor within the state machine.
- bme280_result_t state_machine_bme280_compensate_humidity (ether_t *ether)

  Compensate humidity measurement from the BME280 sensor within the state machine.
- bme280_result_t state_machine_bme280_compensate_temperature (ether_t *ether)

  Compensate temperature measurement from the BME280 sensor within the state machine.
- bme280_result_t state_machine_bme280_compensate_pressure (ether_t *ether)

  Compensate pressure measurement from the BME280 sensor within the state machine.

4.11.1 Function Documentation

4.11.1.1 state_machine_bme280_compensate_humidity()

```
bme280_result_t state_machine_bme280_compensate_humidity (
            ether_t * ether)
```

Compensate humidity measurement from the BME280 sensor within the state machine.

Parameters

out	*ether*	Pointer to the ether structure.

Returns

Result of the humidity compensation operation.

4.11.1.2 state_machine_bme280_compensate_pressure()

```
bme280_result_t state_machine_bme280_compensate_pressure (
            ether_t * ether)
```

Compensate pressure measurement from the BME280 sensor within the state machine.

Parameters

out	*ether*	Pointer to the ether structure.

Returns

Result of the pressure compensation operation.

4.11.1.3 state_machine_bme280_compensate_temperature()

```
bme280_result_t state_machine_bme280_compensate_temperature (
            ether_t * ether)
```

Compensate temperature measurement from the BME280 sensor within the state machine.

Parameters

out	*ether*	Pointer to the ether structure.

Returns

Result of the temperature compensation operation.

4.11.1.4 state_machine_bme280_force_mode()

```
bme280_result_t state_machine_bme280_force_mode (
            ether_t * ether)
```

Set the BME280 sensor to force mode within the state machine.

Parameters

out	*ether*	Pointer to the ether structure.

Returns

Result of the force mode operation.

4.11.1.5 state_machine_bme280_get_compensation_data()

```
bme280_result_t state_machine_bme280_get_compensation_data (
            ether_t * ether)
```

Get compensation data from the BME280 sensor within the state machine.

Parameters

out	*ether*	Pointer to the ether structure.

Returns

Result of the get compensation data operation.

4.11.1.6 state_machine_bme280_id()

```
bme280_result_t state_machine_bme280_id (
            ether_t * ether)
```

Read the ID of the BME280 sensor within the state machine.

Parameters

out	*ether*	Pointer to the ether structure.

Returns

Result of the ID read operation.

4.11.1.7 state_machine_bme280_init()

```
bme280_result_t state_machine_bme280_init (
            ether_t * ether)
```

Initialize the BME280 sensor within the state machine.

Parameters

out	*ether*	Pointer to the ether structure.

Returns

Result of the initialization operation.

4.11.1.8 state_machine_bme280_measure_humidity()

```
bme280_result_t state_machine_bme280_measure_humidity (
            ether_t * ether)
```

Measure humidity with the BME280 sensor within the state machine.

Parameters

out	*ether*	Pointer to the ether structure.

Returns

Result of the humidity measurement operation.

4.11.1.9 state_machine_bme280_measure_pressure()

```
bme280_result_t state_machine_bme280_measure_pressure (
            ether_t * ether)
```

Measure pressure with the BME280 sensor within the state machine.

Parameters

out	*ether*	Pointer to the ether structure.

Returns

Result of the pressure measurement operation.

4.11.1.10 state_machine_bme280_measure_temperature()

```
bme280_result_t state_machine_bme280_measure_temperature (
            ether_t * ether)
```

Measure temperature with the BME280 sensor within the state machine.

Parameters

out	*ether*	Pointer to the ether structure.

Returns

Result of the temperature measurement operation.

4.11.1.11 state_machine_bme280_reset()

```
bme280_result_t state_machine_bme280_reset (
            ether_t * ether)
```

Reset the BME280 sensor within the state machine.

Parameters

out	*ether*	Pointer to the ether structure.

Returns

Result of the reset operation.

4.12 state_machine.h

Go to the documentation of this file.

```
00001 #ifndef INC_STATE_MACHINE_H
00002 #define INC_STATE_MACHINE_H
00003
00004 #include "ether.h"
00005
00012 bme280_result_t state_machine_bme280_init(ether_t *ether);
00013
00020 bme280_result_t state_machine_bme280_reset(ether_t *ether);
00021
00028 bme280_result_t state_machine_bme280_id(ether_t *ether);
00029
00036 bme280_result_t state_machine_bme280_force_mode(ether_t *ether);
00037
00044 bme280_result_t state_machine_bme280_measure_humidity(ether_t *ether);
00045
00052 bme280_result_t state_machine_bme280_measure_temperature(ether_t *ether);
00053
00060 bme280_result_t state_machine_bme280_measure_pressure(ether_t *ether);
00061
00068 bme280_result_t state_machine_bme280_get_compensation_data(ether_t *ether);
00069
00076 bme280_result_t state_machine_bme280_compensate_humidity(ether_t *ether);
00077
00084 bme280_result_t state_machine_bme280_compensate_temperature(ether_t *ether);
00085
00092 bme280_result_t state_machine_bme280_compensate_pressure(ether_t *ether);
00093
00094 #endif // !INC_STATE_MACHINE_H
```

4.13 uart_controller.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include "driver/uart.h"
#include "hal/gpio_types.h"
#include "hal/uart_types.h"
```

Data Structures

- struct uart_controller_descriptor_t

  Structure for UART controller descriptor.

Macros

- #define UART_CONTROLLER_RX_BUF_SIZE (1024)
- #define UART_CONTROLLER_TX_PIN (GPIO_NUM_17)
- #define UART_CONTROLLER_RX_PIN (GPIO_NUM_16)
- #define UART_CONTROLLER_CONFIG_DEFAULT

  Default configuration for the UART controller.
- #define UART_CONTROLLER_DESCRIPTOR_DEFAULT

  Default descriptor for the UART controller.

Enumerations

- enum uart_controller_result_t { UART_CONTROLLER_RESULT_SUCCESS = 0 , UART_CONTROLLER_RESULT_ERROR }

  Result codes for UART controller operations.

Functions

- uart_controller_result_t uart_controller_init (const uart_controller_descriptor_t *uart_controller_descriptor)

  Initialize the UART controller.

4.13.1 Macro Definition Documentation

4.13.1.1 UART_CONTROLLER_CONFIG_DEFAULT

```
#define UART_CONTROLLER_CONFIG_DEFAULT
```

Value:

```
  { \
  .baud_rate = 9600,                      \
  .data_bits = UART_DATA_8_BITS,          \
  .parity = UART_PARITY_DISABLE,          \
  .stop_bits = UART_STOP_BITS_1,          \
  .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,  \
  .source_clk = UART_SCLK_DEFAULT,        \
}
```

Default configuration for the UART controller.

4.13.1.2 UART_CONTROLLER_DESCRIPTOR_DEFAULT

```
#define UART_CONTROLLER_DESCRIPTOR_DEFAULT
```

Value:

```
  {     \
  .uart_config = UART_CONTROLLER_CONFIG_DEFAULT,  \
  .uart_port = UART_NUM_2,                        \
}
```

Default descriptor for the UART controller.

4.13.1.3 UART_CONTROLLER_RX_BUF_SIZE

```
#define UART_CONTROLLER_RX_BUF_SIZE (1024)
```

4.13.1.4 UART_CONTROLLER_RX_PIN

```
#define UART_CONTROLLER_RX_PIN (GPIO_NUM_16)
```

4.13.1.5 UART_CONTROLLER_TX_PIN

```
#define UART_CONTROLLER_TX_PIN (GPIO_NUM_17)
```

4.13.2 Enumeration Type Documentation

4.13.2.1 uart_controller_result_t

```
enum uart_controller_result_t
```

Result codes for UART controller operations.

Enumerator

UART_CONTROLLER_RESULT_SUCCESS	Operation was successful.
UART_CONTROLLER_RESULT_ERROR	Operation encountered an error.

4.13.3 Function Documentation

4.13.3.1 uart_controller_init()

```
uart_controller_result_t uart_controller_init (
            const uart_controller_descriptor_t * uart_controller_descriptor)
```

Initialize the UART controller.

Parameters

in	*uart_controller_descriptor*	Pointer to the UART controller descriptor.

Returns

Result of the initialization operation.

4.14 uart_controller.h

Go to the documentation of this file.

```
00001 #ifndef INC_UART_CONTROLLER_H
00002 #define INC_UART_CONTROLLER_H
00003
00004 #include <stdint.h>
00005 #include <stddef.h>
00006 #include "driver/uart.h"
00007 #include "hal/gpio_types.h"
00008 #include "hal/uart_types.h"
00009
00010 #define UART_CONTROLLER_RX_BUF_SIZE (1024)
00011 #define UART_CONTROLLER_TX_PIN      (GPIO_NUM_17)
00012 #define UART_CONTROLLER_RX_PIN      (GPIO_NUM_16)
00013
00017 typedef enum {
00018   UART_CONTROLLER_RESULT_SUCCESS = 0,
00019   UART_CONTROLLER_RESULT_ERROR,
00020 } uart_controller_result_t;
00021
00025 typedef struct {
00026   uart_config_t uart_config;
00027   uart_port_t uart_port;
00028 } uart_controller_descriptor_t;
00029
00033 #define UART_CONTROLLER_CONFIG_DEFAULT  { \
00034   .baud_rate = 9600,                      \
00035   .data_bits = UART_DATA_8_BITS,          \
00036   .parity = UART_PARITY_DISABLE,          \
00037   .stop_bits = UART_STOP_BITS_1,          \
00038   .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,  \
00039   .source_clk = UART_SCLK_DEFAULT,        \
00040 }
00041
00045 #define UART_CONTROLLER_DESCRIPTOR_DEFAULT  {     \
00046   .uart_config = UART_CONTROLLER_CONFIG_DEFAULT,  \
00047   .uart_port = UART_NUM_2,                        \
00048 }
00049
00056 uart_controller_result_t uart_controller_init(const uart_controller_descriptor_t
      *uart_controller_descriptor);
00057
00058 #endif // !INC_UART_CONTROLLER_H
```

4.15 wifi_controller.h File Reference

```
#include <stdint.h>
#include "esp_system.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_log.h"
#include "esp_wifi_types.h"
#include "nvs_flash.h"
#include "lwip/err.h"
#include "lwip/sys.h"
```

Data Structures

- struct wifi_controller_descriptor_t

  Structure for WIFI controller descriptor.

Macros

- #define WIFI_CONTROLLER_BIT_CONNECTED (BIT0)
- #define WIFI_CONTROLLER_BIT_FAIL (BIT1)
- #define WIFI_CONTROLLER_RETRY_MAX (0x05)

Typedefs

- typedef void(∗ wifi_controller_event_handler_t) (void ∗, esp_event_base_t, int32_t, void ∗)

  Function pointer type for WIFI event handler.

Enumerations

- enum wifi_controller_result_t { WIFI_CONTROLLER_RESULT_SUCCESS = 0 , WIFI_CONTROLLER_RESULT_ERROR }

  Result codes for WIFI controller operations.

Functions

- void wifi_controller_event_handler (void ∗arg, esp_event_base_t event_base, int32_t event_id, void ∗event_data)

  Default WiFi controller configuration and descriptor if SSID and password are defined.
- wifi_controller_result_t wifi_controller_init (const wifi_controller_descriptor_t ∗wifi_controller_descriptor)

  Initialize the WiFi controller.

4.15.1 Macro Definition Documentation

4.15.1.1 WIFI_CONTROLLER_BIT_CONNECTED

```
#define WIFI_CONTROLLER_BIT_CONNECTED (BIT0)
```

4.15.1.2 WIFI_CONTROLLER_BIT_FAIL

```
#define WIFI_CONTROLLER_BIT_FAIL (BIT1)
```

4.15.1.3 WIFI_CONTROLLER_RETRY_MAX

```
#define WIFI_CONTROLLER_RETRY_MAX (0x05)
```

4.15.2 Typedef Documentation

4.15.2.1 wifi_controller_event_handler_t

```
typedef void(* wifi_controller_event_handler_t) (void *, esp_event_base_t, int32_t, void *)
```

Function pointer type for WIFI event handler.

Parameters

in	*arg*	Pointer to user-defined data.
in	*event_base*	Base ID of the event.
in	*event_id*	ID of the event.
in	*event_data*	Pointer to event-specific data.

4.15.3 Enumeration Type Documentation

4.15.3.1 wifi_controller_result_t

```
enum wifi_controller_result_t
```

Result codes for WIFI controller operations.

Enumerator

WIFI_CONTROLLER_RESULT_SUCCESS	Operation was successful.
WIFI_CONTROLLER_RESULT_ERROR	Operation encountered an error.

4.15.4 Function Documentation

4.15.4.1 wifi_controller_event_handler()

```
void wifi_controller_event_handler (
            void * arg,
            esp_event_base_t event_base,
            int32_t event_id,
            void * event_data)
```

Default WiFi controller configuration and descriptor if SSID and password are defined.

Event handler for WiFi events.

Parameters

in	*arg*	Pointer to user-defined data.
in	*event_base*	Base ID of the event.
in	*event_id*	ID of the event.
in	*event_data*	Pointer to event-specific data.

4.15.4.2 wifi_controller_init()

```
wifi_controller_result_t wifi_controller_init (
            const wifi_controller_descriptor_t * wifi_controller_descriptor)
```

Initialize the WiFi controller.

Parameters

in	*wifi_controller_descriptor*	Pointer to the WiFi controller descriptor.

Returns

Result of the initialization operation.

4.16 wifi_controller.h

Go to the documentation of this file.

```
00001 #ifndef INC_WIFI_CONTROLLER_H
00002 #define INC_WIFI_CONTROLLER_H
00003
00004 #include <stdint.h>
00005
00006 #include "esp_system.h"
00007 #include "esp_wifi.h"
00008 #include "esp_event.h"
00009 #include "esp_log.h"
00010 #include "esp_wifi_types.h"
00011 #include "nvs_flash.h"
00012
00013 #include "lwip/err.h"
00014 #include "lwip/sys.h"
00015
00016 #define WIFI_CONTROLLER_BIT_CONNECTED (BIT0)
00017 #define WIFI_CONTROLLER_BIT_FAIL      (BIT1)
00018
00019 #define WIFI_CONTROLLER_RETRY_MAX (0x05)
00020
00024 typedef enum {
00025   WIFI_CONTROLLER_RESULT_SUCCESS = 0,
00026   WIFI_CONTROLLER_RESULT_ERROR,
00027 } wifi_controller_result_t;
00028
00037 typedef void (*wifi_controller_event_handler_t)(void *, esp_event_base_t, int32_t, void *);
00038
00042 typedef struct {
00043   wifi_config_t wifi_config;
00044   wifi_controller_event_handler_t event_handler;
00045 } wifi_controller_descriptor_t;
00046
00050 #if defined (WIFI_CONTROLLER_SETTINGS_SSID) && defined (WIFI_CONTROLLER_SETTINGS_PASSWORD)
00051   #define WIFI_CONTROLLER_CONFIG_DEFAULT  {            \
00052     .sta = {                                          \
00053       .ssid = WIFI_CONTROLLER_SETTINGS_SSID,          \
00054       .password = WIFI_CONTROLLER_SETTINGS_PASSWORD,  \
00055     },                                                \
00056   }
00057
00058   #define WIFI_CONTROLLER_DESCRIPTOR_DEFAULT  {       \
00059     .wifi_config = WIFI_CONTROLLER_CONFIG_DEFAULT,    \
00060     .event_handler = wifi_controller_event_handler,   \
00061   }
00062 #endif
00063
00072 void wifi_controller_event_handler(void *arg, esp_event_base_t event_base,
00073                                    int32_t event_id, void *event_data);
00074
00081 wifi_controller_result_t wifi_controller_init(const wifi_controller_descriptor_t
      *wifi_controller_descriptor);
00082
00083 #endif // !INC_WIFI_CONTROLLER_H
```

Index