AS7352 EVK Firmware

API Documentation
revision v1.1.2

Generated by Doxygen

Contents

1 AS7352 Firmware

1.1 Introduction

This document provides an overview on how the spectral sensor AS7352 can be controlled by a host PC.

The API provides direct access to the AS7352 Chip Library.

Key features:

- Usage of ams standard software components
- Low-level access to Chip Library
- Asynchronous sending of measurement results

The documentation is split into two sections:

- The overview section describes which modules are used and how they interact.
- The module section lists all functions and parameters which can be used.

1.2 Overview

This firmware provides low-level access to the AS7352 Chip Library. For easy code reuse and to reduce the integration effort for customers to a minimum, the interface is kept simple.

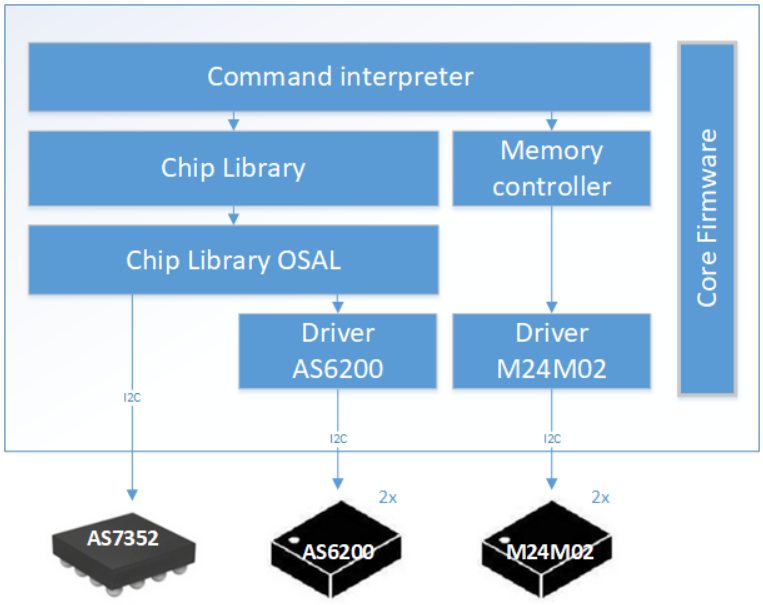

Figure 1 Structure Overview

- *Command interpreter:* All functions of the Chip Library can be called directly via USB
- *Chip Library:* Chip Library for AS7352 which encapsulates all I2C register interactions to a simple interface
- *OSAL:* Operating System Abstraction Layer of the Chip Library which moves all platform-dependent functions to a separate layer, which can be exchanged without adaption of the application
- *Core Firmware:* Firmware base which provides lots of features like I2C/SPI interfaces, system initialisation and USB communication
- *Memory controller:* Manages the write and read requests to and from the EEPROM
- *Driver M24M02:* Driver for EEPROM M24M02
- *Driver AS6200:* Driver for temperature sensor AS6200

1.3 Acronyms and Abbreviations

1.3.1 Acronyms

API = Application Programming Interface
I2C = Inter-Integrated Circuit
FIFO = First in, first out
OSAL = Operating System Abstraction Layer
USB = Universal Serial Bus

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 Module Documentation

4.1 AS7352 Chip Library USB Commands

Description of the application dependent messages.

Data Structures

- struct cmd_chiplib_sync_pwm_t

Enumerations

- enum E_CMD_ID {
  CMD_ID_CHIPLIB_INITIALIZE = 100,
  CMD_ID_CHIPLIB_SHUTDOWN = 101,
  CMD_ID_CHIPLIB_SET_ITEM = 102,
  CMD_ID_CHIPLIB_GET_ITEM = 103,
  CMD_ID_CHIPLIB_SET_CONFIG = 104,
  CMD_ID_CHIPLIB_GET_CONFIG = 105,
  CMD_ID_CHIPLIB_START = 106,
  CMD_ID_CHIPLIB_ABORT = 107,
  CMD_ID_CHIPLIB_CALLBACK = 108,
  CMD_ID_CHIPLIB_STATE = 109,
  CMD_ID_CHIPLIB_SET_EXT_START_SYNC = 121,
  CMD_ID_CHIPLIB_SYNC_PWM = 122 }

Functions

- const prt_table_entry_t * cmd_chiplib_get_table (uint32_t *p_num)

  Returns the supported command table and entry number.
- void cmd_loop (void)

  Cyclic function which shall be called in the main loop.

4.1.1 Detailed Description

Description of the application dependent messages.

These messages will be used to control the measurements remotely. Supports direct control of the chip library.

4.1.2 Enumeration Type Documentation

4.1.2.1 E_CMD_ID `enum E_CMD_ID`

Supported Chip-Lib message IDs

Enumerator

CMD_ID_CHIPLIB_INITIALIZE	This command calls the function as7352_initialize directly. **Input** [Payload-Size: 0 - ...]: • *interface_description:* Optional parameter which will be commited to the OSAL of the ChipLib. **Output** [Payload-Size: 0 byte]
CMD_ID_CHIPLIB_SHUTDOWN	This command calls the function as7352_shutdown directly. **Input** [Payload-Size: 0 byte] **Output** [Payload-Size: 0 byte]
CMD_ID_CHIPLIB_SET_ITEM	This command calls the function as7352_set_item directly. **Input** [Payload-Size: 1 - ...]: • *item_id:* Size uint8. Identification number of an item, see as7352_item_ids. • *item_data:* Size uint8 * (payload_size - 1). Data of the item. **Output** [Payload-Size: 0 byte]

Enumerator

CMD_ID_CHIPLIB_GET_ITEM	This command calls the function as7352_get_item directly. **Input** [Payload-Size: 2 byte]: • *item_id:* Size uint8. Identification number of an item, see as7352_item_ids. • *item_size:* Size uint8. Size in byte of data, see as7352_item_sizes. **Output** [Payload-Size: ...]: • *item_data:* Size uint8 ∗ (item_size). Data of the item. definition
CMD_ID_CHIPLIB_SET_CONFIG	This command calls the function as7352_set_configuration directly. **Input** [Payload-Size: 0 - ...]: • *config_data:* Size uint8 ∗ (payload_size) See parameter definition of as7352_set_configuration **Output** [Payload-Size: 0 byte]
CMD_ID_CHIPLIB_GET_CONFIG	This command calls the function as7352_get_configuration directly. **Input** [Payload-Size: 4 byte]: • *config_data_size:* Size uint32. Maximum size for configuration data **Output** [Payload-Size: ...]: • *config_data:* Size uint8 ∗ (Payload-Size). For parameter format see as7352_get_configuration definition
CMD_ID_CHIPLIB_START	This command calls the function as7352_start_measurement directly. **Input** [Payload-Size: 0 byte] **Output** [Payload-Size: 0 byte]

Enumerator

CMD_ID_CHIPLIB_ABORT	This command calls the function as7352_abort_measurement directly. **Input** [Payload-Size: 0 byte] **Output** [Payload-Size: 0 byte]
CMD_ID_CHIPLIB_CALLBACK	It will be used to send measurement data acyclic. **Note** ID is not supported for a host request! **Input** [Payload-Size: 0 byte] **Output** [Payload-Size: 0 - ...] • *data_size:* Size uint16. Size of measurement data • *data:* Size uint8 ∗ (data_size). Measurement data • *items:* Size uint8 ∗ (Payload-Size - data_size - 2). Item data
CMD_ID_CHIPLIB_STATE	It will be used to send measurement state acyclic. **Note** ID is not supported for a host request! **Input** [Payload-Size: 0 byte] **Output** [Payload-Size: 2] • *state:* Size uint8. See as7352_states • *result:* Size uint8. Return value of function as7352_execute_state_machine
CMD_ID_CHIPLIB_SET_EXT_START_SYNC	if enabled, an one millisecond low pulse will be set on GPIO 3, when command with id CMD_ID_CHIPLIB_START will be called. **Note** If disabled (default state) GPIO 3 is configured as floating pin. **Input** [Payload-Size: 1 byte] • *enable:* Size uint8. 0 < a low pulse will be set on command id CMD_ID_CHIPLIB_START **Output** [Payload-Size: 0 byte]

Enumerator

CMD_ID_CHIPLIB_SYNC_PWM	Controls the Sync PWM to generate pulses for time-resolved measurement. **Input** [Payload-Size: 20 bytes] • See structure cmd_chiplib_sync_pwm_t. **Output** [Payload-Size: 0 byte]

4.1.3 Function Documentation

4.1.3.1 cmd_chiplib_get_table() `const prt_table_entry_t* cmd_chiplib_get_table ( uint32_t * p_num )`

Returns the supported command table and entry number.

This table defines the callback functions, which will be used by the protcol handler

Parameters

in	*p_num*	- memory, where the number of table elements can be saved

Returns

Address which points to the command table

4.1.3.2 cmd_loop() `void cmd_loop ( void )`

Cyclic function which shall be called in the main loop.

This function is not necessary for handling the command table. Only some specific cyclic functions will be called here, like temperature measurement and status update

4.2 Core-Firmware USB Commands

Handles the command IDs and the corresponding functions.

Data Structures

- struct i2c_spi_config
- struct i2c_xfer
- struct i2c_xfer_16bit
- struct pio_config
- struct pwm_config
- struct test_req

Typedefs

- typedef struct i2c_spi_config i2c_spi_config_t
- typedef struct i2c_xfer i2c_xfer_t
- typedef struct i2c_xfer_16bit i2c_xfer_16bit_t
- typedef struct pio_config pio_config_t
- typedef struct pwm_config pwm_config_t
- typedef struct test_req test_req_t

Enumerations

- enum E_CMD_BASE_ID {
  CMD_BASE_ID_APPL_NAME = 0,
  CMD_BASE_ID_VERSION = 1,
  CMD_BASE_ID_RESET = 2,
  CMD_BASE_ID_I2C_CONFIG = 3,
  CMD_BASE_ID_I2C_XFER = 4,
  CMD_BASE_ID_SPI_CONFIG = 5,
  CMD_BASE_ID_SPI_XFER = 6,
  CMD_BASE_ID_PIO_CONFIG = 7,
  CMD_BASE_ID_PIO_XFER = 8,
  CMD_BASE_ID_PIO_STATE = 9,
  CMD_BASE_ID_SYS_START_BL = 10,
  CMD_BASE_ID_PWM_CONFIG = 11,
  CMD_BASE_ID_TEST_REQ = 12,
  CMD_BASE_ID_TEST_RSP = 13,
  CMD_BASE_ID_I2C_XFER_16BIT = 14,
  CMD_BASE_ID_HW_REV = 15 }

Functions

- const prt_table_entry_t * cmd_base_get_table (uint32_t *p_num)

  Returns the supported command table and entry number.
- void cmd_main_loop (void)

  Handle ongoing command related tasks.

4.2.1 Detailed Description

Handles the command IDs and the corresponding functions.

- Managing of the base command IDs
- Definition of the message structures
- Definition of the message functions

4.2.2 Typedef Documentation

4.2.2.1 i2c_spi_config_t `typedef struct i2c_spi_config i2c_spi_config_t`

Configuration structure for I2C and SPI periphery

4.2.2.2 i2c_xfer_t `typedef struct i2c_xfer i2c_xfer_t`

I2C transfer data

4.2.2.3 i2c_xfer_16bit_t `typedef struct i2c_xfer_16bit i2c_xfer_16bit_t`

I2C transfer data

4.2.2.4 pio_config_t `typedef struct pio_config pio_config_t`

Configuration structure for the initialization of the GPIO module

4.2.2.5 pwm_config_t `typedef struct pwm_config pwm_config_t`

Configuration structure for the initialization of the PWM pin

4.2.2.6 test_req_t `typedef struct test_req test_req_t`

Structure for test message stream request

4.2.3 Enumeration Type Documentation

4.2.3.1 E_CMD_BASE_ID `enum E_CMD_BASE_ID`

Identifiers for the base commands which shall be supported by all core devices

Enumerator

CMD_BASE_ID_APPL_NAME	This command returns the name of the application. **Input** [Payload-Size: 0] **Output** [Payload-Size: 0 - 40 byte]: • *application_name:* Name of the application
CMD_BASE_ID_VERSION	This command returns the firmware version. **Input** [Payload-Size: 0] **Output** [Payload-Size: 0 - 20 byte]: • *firmware_version:* Version string like 1.0.0
CMD_BASE_ID_RESET	This command triggers the firmware to restart **Attention** On success, no acknowledge will be send to the host **Input** [Payload-Size: 0] **Output** [Payload-Size: 0]
CMD_BASE_ID_I2C_CONFIG	This command initialize the I2C interface **Note** Target-id will be used to select the channel. **Input** [Payload-Size: 8]: • *enable:* Size uint8. 1 - interface will be enabled • *reserved:* Size uint8[3]. Reserved for future usage • *frequency:* Size uint32. I2C bus frequency in Hz (eg. 100kHz, 400kHz) **Output** [Payload-Size: 0 byte]

Enumerator

CMD_BASE_ID_I2C_XFER	Transfer of an I2C datagram. **Attention** Deprecated since version 3.0.0, use CMD_BASE_ID_I2C_XFER_16BIT instead. **Note** Target-id will be used to select the channel. Structure i2c_xfer_t can be used for header initialization **Input** [Payload-Size: 2 - 257]: • *address:* Size uint8. I2C slave device address • *recv_size:* Size uint8. Number of data which shall be read after sent data • *send_data:* Optional: Size uint8[0..255]. Content of send data **Output** [Payload-Size: 0 byte]
CMD_BASE_ID_SPI_CONFIG	This command initialize the SPI interface **Note** Target-id will be used to select the channel. **Input** [Payload-Size: 8]: • *enable:* Size uint8. 1 - interface will be enabled • *mode:* Size uint8. See enumeration spi_modes • *first_bit:* Size uint8. See enumeration spi_first_bit • *reserved:* Size uint8. Reserved for future usage • *frequency:* Size uint32. SPI bus frequency in Hz (eg. 1MHz) **Output** [Payload-Size: 0 byte]

Enumerator

CMD_BASE_ID_SPI_XFER	Transfer of a SPI datagram **Note** Target-id will be used to select the channel. **Input** [Payload-Size: 1 - 65535]: • *send_data:* Size uint8[1..65535]. Content of send data **Output** [Payload-Size: 0 byte] • *recv_data:* Size of send_data. Content of recv data
CMD_BASE_ID_PIO_CONFIG	This command initialize the PIO (Pin Input Output) interface **Note** Target-id will be used to select the channel. **Input** [Payload-Size: 3]: • *enable:* Size uint8. 1 - pin will be enabled • *mode:* Size uint8. See enumeration pio_modes • *pull:* Size uint8. See enumeration pio_pulls **Output** [Payload-Size: 0 byte]
CMD_BASE_ID_PIO_XFER	Set/Get the state of a pin **Note** Target-id will be used to select the channel. **Input** [Payload-Size: 0 - 1]: • *state:* Optional: Size uint8. New pin state. See pio_states_t **Output** [Payload-Size: 1 byte]: • *state:* Size uint8. Current pin state. See pio_states_t

Enumerator

CMD_BASE_ID_PIO_STATE	Asynchronous message ID for GPIO state **Note** Target-id will be used to select the channel. **Attention** This command ID is not supported directly. It will be used to send cyclic pin change-states (if registered). **Input** [Payload-Size: 0] **Output** [Payload-Size: 1 byte] • *state:* Size uint8. Current pin state. See pio_states_t
CMD_BASE_ID_SYS_START_BL	This command triggers the firmware to start the bootloader **Attention** On success, no acknowledge will be send to the host **Input** [Payload-Size: 0] **Output** [Payload-Size: 0]
CMD_BASE_ID_PWM_CONFIG	This command initialize a PWM channel **Note** Target-id will be used to select the channel. **Input** [Payload-Size: 8]: • *enable:* Size uint8. 1 - interface will be enabled • *reserved:* Size uint8. Reserved for future usage • *duty_cycle:* Size uint16. duty cycle [0 - 1000] • *frequency:* Size uint32. PWM freqency in hertz **Output** [Payload-Size: 0 byte]

Enumerator

CMD_BASE_ID_TEST_REQ	This command requests a stream of test messages **Input** [Payload-Size: 8]: • *count:* Size uint16. Number of test messages to send • *size:* Size uint16. Size of test message payload • *delay_us:* Size uint32. Minimum time between the transmission of two test messages in microseconds **Output** [Payload-Size: 0 byte]
CMD_BASE_ID_TEST_RSP	This command is used to stream test data requested with CMD_BASE_ID_TEST_REQ **Output** [Payload-Size: variable] • *counter:* Size uint16. Number of remaining test messages, not including the current message. Field is only present if requested payload size is larger than the size of this field. • *pattern:* All payload bytes not used by other fields contain their index in the payload (modulo 255), i.e. payload bytes at indices 2 and 258 both have the value 2.
CMD_BASE_ID_I2C_XFER_16BIT	Transfer of an I2C datagram with extended sizes of more than 8bit **Note** Target-id will be used to select the channel. Structure i2c_xfer_16bit_t can be used for header initialization **Input** [Payload-Size: 4 - ...]: • *address:* Size uint8. I2C slave device address • *reserved:* Size uint8. not used, set to zero (padding byte) • *recv_size:* Size uint16. Number of bytes which shall be read after sent data (Can be zero) • *send_data:* Optional: Bytes which shall be sent before data will be read **Output** [Payload-Size: 0 - ...] • *recv_data:* Optional: Received bytes

Enumerator

CMD_BASE_ID_HW_REV	This command returns the hardware revision. **Input** [Payload-Size: 0] **Output** [Payload-Size: 0 - 40 byte]: • *hw_rev:* Hardware revision of the target

4.2.4 Function Documentation

4.2.4.1 cmd_base_get_table() `const prt_table_entry_t* cmd_base_get_table ( uint32_t * p_num )`

Returns the supported command table and entry number.

This table defines the callback functions, which will be used by the protcol handler

Parameters

in	*p_num*	memory, where the number of table elements can be saved

Returns

Address which points to the command table

4.2.4.2 cmd_main_loop() `void cmd_main_loop ( void )`

Handle ongoing command related tasks.

This function should be called on every main loop iteration.

4.3 AS7352 Chip Library Definitions

Description of the used data types.

Data Structures

- struct as7352_serial
- struct as7352_version
- struct as7352_led_pattern
- struct as7352_led_config
- struct as7352_auto_gain

Macros

- #define CHIP_LIB_IDENT 7352
- #define AS7352_LED_PATTERN_NUM 10
- #define AS7352_MAX_ITEM_BUFFER_SIZE 80
- #define AS7352_MAX_DATA_BUFFER_SIZE 256
- #define AS7352_GAIN_FACTOR_NUM 15
- #define AS7352_SATURATED 65535

Typedefs

- typedef void(∗ as7352_callback_t) (uint8_t device, uint8_t error, void ∗p_data, uint32_t data_size, void ∗p_↩ items, uint32_t items_size, void ∗p_cb_param)

  Callback function, which transfers the measurement results to the application.

Enumerations

- enum as7352_saturation_flags {
  SATURATION_FLAG_FD_DIGITAL = 0x01,
  SATURATION_FLAG_FD_ANALOG = 0x02,
  SATURATION_FLAG_SPECTRAL_ANALOG = 0x08,
  SATURATION_FLAG_SPECTRAL_DIGITAL = 0x10 }
- enum as7352_measurement_types {
  MEASUREMENT_TYPE_SPECTRAL = 0,
  MEASUREMENT_TYPE_FIFO = 1,
  MEASUREMENT_TYPE_SPECTRAL_FIFO = 2,
  MEASUREMENT_TYPE_GOERTZEL = 3,
  MEASUREMENT_TYPE_NUM = 4 }

- enum as7352_channels {
  CHANNEL_DISABLED = 0,
  CHANNEL_F1 = 1,
  CHANNEL_F2 = 2,
  CHANNEL_FZ = 3,
  CHANNEL_F3 = 4,
  CHANNEL_F4 = 5,
  CHANNEL_FY = 6,
  CHANNEL_F5 = 7,
  CHANNEL_FXL = 8,
  CHANNEL_F6 = 9,
  CHANNEL_F7 = 10,
  CHANNEL_F8 = 11,
  CHANNEL_NIR = 12,
  CHANNEL_FD = 13,
  CHANNEL_DARK = 14,
  CHANNEL_VISLT = 16,
  CHANNEL_VISRB = 32,
  CHANNEL_VISRT = 64,
  CHANNEL_VISLB = 128 }
- enum as7352_gain {
  GAIN_0_5X = 0,
  GAIN_1X = 1,
  GAIN_2X = 2,
  GAIN_4X = 3,
  GAIN_8X = 4,
  GAIN_16X = 5,
  GAIN_32X = 6,
  GAIN_64X = 7,
  GAIN_128X = 8,
  GAIN_256X = 9,
  GAIN_512X = 10,
  GAIN_1024X = 11,
  GAIN_2048X = 12,
  GAIN_4096X = 13,
  GAIN_5120X = 14 }
- enum as7352_led_masks {
  LED_MASK_OFF = 0x00,
  LED_MASK_INTERN = 0x01,
  LED_MASK_EXT_0 = 0x02,
  LED_MASK_EXT_1 = 0x04,
  LED_MASK_EXT_2 = 0x08,
  LED_MASK_EXT_3 = 0x10,
  LED_MASK_EXT_4 = 0x20,
  LED_MASK_EXT_5 = 0x40,
  LED_MASK_OUTPUT = 0x80 }
- enum as7352_sync_mode {
  SYNC_MODE_DISABLED = 0,
  SYNC_MODE_RISING_EDGE = 1,
  SYNC_MODE_FALLING_EDGE = 2,
  SYNC_MODE_NUMBER = 3 }
- enum as7352_item_ids {
  ITEM_ID_RESERVED = 0,
  ITEM_ID_ASTEP = 1,

ITEM_ID_ATIME = 2,
ITEM_ID_ITIME = 3,
ITEM_ID_AGAIN = 4,
ITEM_ID_MEAS_TYPE = 5,
ITEM_ID_BREAK = 6,
ITEM_ID_CHANNELS = 7,
ITEM_ID_VERSION = 8,
ITEM_ID_SERIAL = 9,
ITEM_ID_AUTOZERO = 10,
ITEM_ID_MEAS_COUNT_ALS = 11,
ITEM_ID_LED_PATTERN = 12,
ITEM_ID_LED_WAIT_TIME = 13,
ITEM_ID_INTERRUPT_PIN = 14,
ITEM_ID_LED_INTERN = 15,
ITEM_ID_LED_EXT_0 = 16,
ITEM_ID_LED_EXT_1 = 17,
ITEM_ID_LED_EXT_2 = 18,
ITEM_ID_LED_EXT_3 = 19,
ITEM_ID_LED_EXT_4 = 20,
ITEM_ID_LED_EXT_5 = 21,
ITEM_ID_OUTPUT = 22,
ITEM_ID_TEMP_EXT_0 = 23,
ITEM_ID_TEMP_EXT_1 = 24,
ITEM_ID_TEMP_EXT_2 = 25,
ITEM_ID_TEMP_EXT_3 = 26,
ITEM_ID_TEMP_EXT_4 = 27,
ITEM_ID_TEMP_EXT_5 = 28,
ITEM_ID_MEASURE_ITEMS = 29,
ITEM_ID_FGAIN = 30,
ITEM_ID_FTIME = 31,
ITEM_ID_FTIME_US = 32,
ITEM_ID_TIMESTAMP = 33,
ITEM_ID_AUTO_GAIN_RANGE_ALS = 34,
ITEM_ID_AUTO_GAIN_RANGE_FIFO = 35,
ITEM_ID_GAIN_FACTORS = 36,
ITEM_ID_REG_READ = 37,
ITEM_ID_SATURATION = 38,
ITEM_ID_SYNC_MODE = 39,
ITEM_ID_FIFO_AGC_BLOCKSIZE = 40,
ITEM_ID_FD_COEFF = 41,
ITEM_ID_FD_DCR = 42,
ITEM_ID_FD_PERS = 43,
ITEM_ID_FD_AGC_DISABLE = 44,
ITEM_ID_FD_MODCLK = 45,
ITEM_ID_FD_SAMPLES = 46,
ITEM_ID_FD_COMPARE_VALUE = 47,
ITEM_ID_MEAS_COUNT_FIFO = 48,
ITEM_ID_MAX = 49 }

- enum as7352_item_sizes {
ITEM_SIZE_RESERVED = 0,
ITEM_SIZE_ASTEP = 2,
ITEM_SIZE_ATIME = 1,
ITEM_SIZE_ITIME = 4,
ITEM_SIZE_AGAIN = 1,

ITEM_SIZE_MEAS_TYPE = 1,
ITEM_SIZE_BREAK = 4,
ITEM_SIZE_CHANNELS_MAX = 18,
ITEM_SIZE_VERSION = 4,
ITEM_SIZE_SERIAL = 8,
ITEM_SIZE_SATURATION = 1,
ITEM_SIZE_AUTOZERO = 1,
ITEM_SIZE_MEAS_COUNT_ALS = 2,
ITEM_SIZE_MEAS_COUNT_FIFO = 4,
ITEM_SIZE_LED_PATTERN = 20,
ITEM_SIZE_LED_WAIT_TIME = 4,
ITEM_SIZE_INTERRUPT_PIN = 1,
ITEM_SIZE_LED_INTERN = 4,
ITEM_SIZE_LED_EXT_0 = 4,
ITEM_SIZE_LED_EXT_1 = 4,
ITEM_SIZE_LED_EXT_2 = 4,
ITEM_SIZE_LED_EXT_3 = 4,
ITEM_SIZE_LED_EXT_4 = 4,
ITEM_SIZE_LED_EXT_5 = 4,
ITEM_SIZE_OUTPUT = 1,
ITEM_SIZE_TEMP_EXT_0 = 4,
ITEM_SIZE_TEMP_EXT_1 = 4,
ITEM_SIZE_TEMP_EXT_2 = 4,
ITEM_SIZE_TEMP_EXT_3 = 4,
ITEM_SIZE_TEMP_EXT_4 = 4,
ITEM_SIZE_TEMP_EXT_5 = 4,
ITEM_SIZE_MEASURE_ITEMS = 10,
ITEM_SIZE_FGAIN = 1,
ITEM_SIZE_FTIME = 2,
ITEM_SIZE_FTIME_US = 4,
ITEM_SIZE_TIMESTAMP = 8,
ITEM_SIZE_AUTO_GAIN_RANGE_ALS = 2,
ITEM_SIZE_AUTO_GAIN_RANGE_FIFO = 2,
ITEM_SIZE_GAIN_FACTORS = 30,
ITEM_SIZE_REG_READ = 1,
ITEM_SIZE_SYNC_MODE = 1,
ITEM_SIZE_FIFO_AGC_BLOCKSIZE = 2,
ITEM_SIZE_FD_COEFF = 1,
ITEM_SIZE_FD_DCR = 1,
ITEM_SIZE_FD_PERS = 1,
ITEM_SIZE_FD_AGC_DISABLE = 1,
ITEM_SIZE_FD_MODCLK = 1,
ITEM_SIZE_FD_SAMPLES = 1,
ITEM_SIZE_FD_COMPARE_VALUE = 1 }

- enum as7352_states {
  STATE_CONFIG = 0,
  STATE_MEASURE = 1 }

4.3.1 Detailed Description

Description of the used data types.

These are the type definitions used by AS7352 chip library.

4.3.2 Macro Definition Documentation

4.3.2.1 CHIP_LIB_IDENT `#define CHIP_LIB_IDENT 7352`

ChipLib identification

4.3.2.2 AS7352_LED_PATTERN_NUM `#define AS7352_LED_PATTERN_NUM 10`

Number of supported LED pattern configuration

4.3.2.3 AS7352_MAX_ITEM_BUFFER_SIZE `#define AS7352_MAX_ITEM_BUFFER_SIZE 80`

Maximum size in bytes of additional item buffer in callback function

4.3.2.4 AS7352_MAX_DATA_BUFFER_SIZE `#define AS7352_MAX_DATA_BUFFER_SIZE 256`

Maximum size in bytes of data buffer in callback function

4.3.2.5 AS7352_GAIN_FACTOR_NUM `#define AS7352_GAIN_FACTOR_NUM 15`

Number of supported gains

4.3.2.6 AS7352_SATURATED `#define AS7352_SATURATED 65535`

Measured ADC value is saturated

4.3.3 Typedef Documentation

4.3.3.1 as7352_callback_t `typedef void(* as7352_callback_t) (uint8_t device, uint8_t error, void *p_data, uint32_t data_size, void *p_items, uint32_t items_size, void *p_cb_param)`

Callback function, which transfers the measurement results to the application.

This callback type will be registered via the function as7352_initialize. During the measurement, this function transfers the cyclic results.

p_data transfers the measured sensor data. p_items transfers additional item content, if configured. The content of the items is without item size and without item id. The order of the item can be readout by ITEM_ID_MEASURE_ITEMS.

See also

meas_data_sec

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes)
in	*error*	Default ERR_SUCCESS, otherwise an error is occurred during measurement and the measurement. stops. See error_codes
in	*p_data*	Pointer to the measurement data, the content depends on configuration. See ITEM_ID_MEAS_TYPE.
in	*data_size*	Size of measurement data in bytes.
in	*p_items*	Returns an optional array with data, which can be configured with item ITEM_ID_MEASURE_ITEMS.
in	*items_size*	Size of item data in bytes. The maximum supported size is defined in definition AS7352_MAX_ITEM_BUFFER_SIZE
in	*p_cb_param*	Application parameter which was defined during call of as7352_initialize.

4.3.4 Enumeration Type Documentation

4.3.4.1 as7352_saturation_flags `enum as7352_saturation_flags`

List of saturation flags ::ITEM_ID_SATURATION

4.3.4.2 as7352_measurement_types `enum as7352_measurement_types`

List of supported measurement types for ITEM_ID_MEAS_TYPE

Enumerator

MEASUREMENT_TYPE_SPECTRAL	Spectral measurement: Measures six chanels in parallel, otherwise you can measure 12 channels in two blocks a 6 chanels (twice integration time!).
MEASUREMENT_TYPE_FIFO	FIFO measurement: Measures with one ADC only in fast FIFO mode: It is possible to map more than one channel to this ADC.
MEASUREMENT_TYPE_SPECTRAL_FIFO	Parallel FIFO and spectral measurement. Combinatation of mode MEASUREMENT_TYPE_SPECTRAL and MEASUREMENT_TYPE_FIFO with less options
MEASUREMENT_TYPE_GOERTZEL	On chip internal flicker detection: Uses internal flicker detection with Goertzel algorithm. Callback payload contains 2 bytes: • byte[0] : flicker frequency • byte[1] : FD_Status register value
MEASUREMENT_TYPE_NUM	Maximum supported measurement types.

4.3.4.3 as7352_channels `enum as7352_channels`

Supported channel types for spectral channels. Used to ITEM_ID_CHANNELS.

Enumerator

CHANNEL_DISABLED	place holder for unused channels
CHANNEL_F1	channel F1
CHANNEL_F2	channel F2
CHANNEL_FZ	channel FZ
CHANNEL_F3	channel F3
CHANNEL_F4	channel F4
CHANNEL_FY	channel FY
CHANNEL_F5	channel F5
CHANNEL_FXL	channel FXL
CHANNEL_F6	channel F6
CHANNEL_F7	channel F7
CHANNEL_F8	channel F8
CHANNEL_NIR	channel NIR
CHANNEL_FD	channel FLICKER
CHANNEL_DARK	channel DARK
CHANNEL_VISLT	channel VISLT
CHANNEL_VISRB	channel VISRB
CHANNEL_VISRT	channel VISRT
CHANNEL_VISLB	channel VISLB

4.3.4.4 as7352_gain `enum as7352_gain`

Supported gains for items ITEM_ID_AGAIN and ITEM_ID_FGAIN to set the spectral sensitivity.

Enumerator

GAIN_0_5X	gain of 0.5x
GAIN_1X	gain of 1x
GAIN_2X	gain of 2x
GAIN_4X	gain of 4x
GAIN_8X	gain of 8x
GAIN_16X	gain of 16x
GAIN_32X	gain of 32x
GAIN_64X	gain of 64x
GAIN_128X	gain of 128x
GAIN_256X	gain of 256x
GAIN_512X	gain of 512x
GAIN_1024X	gain of 1024x
GAIN_2048X	gain of 2048x

Enumerator

GAIN_4096X	gain of 4096x
GAIN_5120X	gain of 5120x

4.3.4.5 as7352_led_masks `enum as7352_led_masks`

This bit masks will be used for configuration of the LED pattern.

See also

ITEM_ID_LED_PATTERN

Enumerator

LED_MASK_OFF	mask that no LED will be set
LED_MASK_INTERN	mask that the internal LED will be set
LED_MASK_EXT_0	mask that the external LED 0 will be set
LED_MASK_EXT_1	mask that the external LED 1 will be set
LED_MASK_EXT_2	mask that the external LED 2 will be set
LED_MASK_EXT_3	mask that the external LED 3 will be set
LED_MASK_EXT_4	mask that the external LED 4 will be set
LED_MASK_EXT_5	mask that the external LED 5 will be set
LED_MASK_OUTPUT	mask that the output pin will be set

4.3.4.6 as7352_sync_mode `enum as7352_sync_mode`

Supported sync modes for ITEM_ID_SYNC_MODE

Enumerator

SYNC_MODE_DISABLED	Trigger measurement via GPIO pin disabled
SYNC_MODE_RISING_EDGE	Trigger measurement on rising edge of signal connected to GPIO pin
SYNC_MODE_FALLING_EDGE	Trigger measurement on falling edge of signal connected to GPIO pin
SYNC_MODE_NUMBER	Number of supported sync modes (not a valid payload value for ITEM_ID_SYNC_MODE)

4.3.4.7 as7352_item_ids `enum as7352_item_ids`

List of all supported item IDs

Enumerator

ITEM_ID_RESERVED	This ID will be used for special cases, like detection of undefined item.

ITEM_ID_ASTEP	This item access directly the register ASTEP 0xCA/0xCB. Those registers are one of the two parts which allows the user to set the integration time for spectral measurement.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_ASTEP (unsigned 16bit)
	• *Parameter range:* 1 - 65534
	• *Default value:* 599
	• *Direction:* write/read
	
	Example
	
	`uint16_t astep_set, astep_get;`
	`// set ASTEP-property`
	`astep_set = 999;`
	`as7352_set_item(DEV_ID, ITEM_ID_ASTEP, (void *)&astep_set,`
	`ITEM_SIZE_ASTEP);`
	`// get ASTEP-property`
	`as7352_get_item(DEV_ID, ITEM_ID_ASTEP, (void *)&astep_get,`
	`ITEM_SIZE_ASTEP);`
	
	Note
	
	If you want to set the integration time in microseconds directly, use ITEM_ID_ITIME. Here you can find additional information as well.

Enumerator

ITEM_ID_ATIME	This item accesses directly the AS7352 ATIME-register 0x81. ATIME is the second part to calculate the integration time.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_ATIME (unsigned 8bit)
	• *Parameter range:* 0 - 255
	• *Default value:* 29
	• *Direction:* write/read
	
	Example
	
	`uint8_t atime_set, atime_get;`
	`// set ATIME-property`
	`atime_set = 19;`
	`as7352_set_item(DEV_ID, ITEM_ID_ATIME, (void *)&atime_set,`
	`ITEM_SIZE_ATIME);`
	`// get ATIME-property`
	`as7352_get_item(DEV_ID, ITEM_ID_ATIME, (void *)&atime_get,`
	`ITEM_SIZE_ATIME);`
	
	Note
	
	If you want to set the integration time in microseconds directly, use ITEM_ID_ITIME. Here you can find additional information as well.

Enumerator

ITEM_ID_ITIME	This item calculates the integration time in microseconds. Internally, the registers ATIME and ASTEP will be set. Following formula describes the relationship:
	
	$t_{int} = (ATIME + 1) * (ASTEP + 1) * (2000/720)us$
	
	The maximum possible ADC fullscale range has a width of 16bit, but is limited by the integration time:
	
	$ADC_{fullscale} = (ATIME + 1) * (ASTEP + 1)$
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_ITIME (unsigned 32bit)
	• *Parameter range:* 6 - 46602667
	• *Default value:* 50000
	• *Direction:* write/read
	
	Example
	
	`uint32_t itime_set, itime_get;`
	`// set ITIME-property`
	`itime_set = 40000;`
	`as7352_set_item(DEV_ID, ITEM_ID_ITIME, (void *)&itime_set,`
	`ITEM_SIZE_ITIME);`
	`// get ITIME-property`
	`as7352_get_item(DEV_ID, ITEM_ID_ITIME, (void *)&itime_get,`
	`ITEM_SIZE_ITIME);`
	
	Attention
	
	It is normal that the value which you readback differs from the set value, because it depends on the resolution of ASTEP and ATIME

Enumerator

ITEM_ID_AGAIN	This item accesses directly the AS7352 CFG_1-register (0xAA) bits AGAIN to change the spectral sensitivity.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_AGAIN (unsigned 8bit)
	• *Parameter range:* see as7352_gain
	• *Default value:* GAIN_256X
	• *Direction:* write/read
	
	Example
	
	`uint8_t again_set, again_get;`
	`// set AGAIN-property`
	`again_set = GAIN_16X;`
	`as7352_set_item(DEV_ID, ITEM_ID_AGAIN, (void *)&again_set,`
	`ITEM_SIZE_AGAIN);`
	`// get AGAIN-property`
	`as7352_get_item(DEV_ID, ITEM_ID_AGAIN, (void *)&again_get,`
	`ITEM_SIZE_AGAIN);`

ITEM_ID_MEAS_TYPE	This item configures the measurement typ. (as7352_measurement_types)
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_MEAS_TYPE (unsigned 8bit)
	• *Parameter range:* see as7352_measurement_types
	• *Default value:* MEASUREMENT_TYPE_SPECTRAL
	• *Direction:* write/read
	
	Example
	
	`uint8_t meas_type_set, meas_type_get;`
	`// set MEAS_TYPE-property`
	`meas_type_set = MEASUREMENT_TYPE_FIFO;`
	`as7352_set_item(DEV_ID, ITEM_ID_MEAS_TYPE, (void`
	`*)&meas_type_set, ITEM_SIZE_MEAS_TYPE);`
	`// get MEAS_TYPE-property`
	`as7352_get_item(DEV_ID, ITEM_ID_MEAS_TYPE, (void`
	`*)&meas_type_get, ITEM_SIZE_MEAS_TYPE);`
	
	See also
	
	as7352_measurement_types

Enumerator

ITEM_ID_BREAK	This item specifies the break between the finished measurement and the start of the next measurement.
	On default this time is disabled and not used. This time depends on the integration time of the sensor because the value will be calculated by the difference of wait_time and integration_time-registers.
	If you want to readout the real configured break time, you must set integration time at first, then set break time itself and afterwards read it back.
	The maximum time of integration is much higher than the maximum possible wait time registers. So, you can't set the break time in all cases. To use this break time, decrease the integration time.
	Note
	The minimum possible break time is 2780us.
	Properties
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_BREAK (unsigned 32bit)
	• *Parameter range:* 0, 2780us - 10000000us
	• *Default value:* 0
	• *Direction:* write/read
	Example
	`uint32_t break_set, break_get;`
	`// set BREAK-property`
	`break_set = 50000; 50ms`
	`as7352_set_item(DEV_ID, ITEM_ID_BREAK, (void *)&break_set, ITEM_SIZE_BREAK);`
	`// get BREAK-property`
	`as7352_get_item(DEV_ID, ITEM_ID_BREAK, (void *)&break_get, ITEM_SIZE_BREAK);`

Enumerator

ITEM_ID_CHANNELS	Configures up to 18 measurement channels, each byte for one channel. The possible values for every byte are described in as7352_channels. **Properties** • *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL • *Payload size:* ::ITEM_SIZE_CHANNELS (unsigned 18byte) • *Parameter range:* as7352_channels • *Default value:* CHANNEL_FZ CHANNEL_FY CHANNEL_FXL, CHANNEL_NIR, ::CHANNEL_CLEAR, ::CHANNEL_FLICKER, CHANNEL_F2, CHANNEL_F3, CHANNEL_F4, CHANNEL_F5, ::CHANNEL_CLEAR, ::CHANNEL_FLICKER, CHANNEL_F1, CHANNEL_F6, CHANNEL_F7, CHANNEL_F8, ::CHANNEL_CLEAR, ::CHANNEL_FLICKER, • *Direction:* write/read **Example** `// configure a single measuremt with 8 channels only uint8_t channels[ITEM_SIZE_CHANNELS_MAX] = {CHANNEL_F1, CHANNEL_F2, CHANNEL_F3, CHANNEL_F4, CHANNEL_F5, CHANNEL_F6, CHANNEL_F7, CHANNEL_F8, CHANNEL_DISABLED, CHANNEL_DISABLED, CHANNEL_DISABLED,CHANNEL_DISABLED, CHANNEL_DISABLED, CHANNEL_DISABLED, CHANNEL_DISABLED, CHANNEL_DISABLED}, CHANNEL_DISABLED, CHANNEL_DISABLED;` `// set CHANNELS-property` `as7352_set_item(DEV_ID, ITEM_ID_CHANNELS, (void *)channels, ITEM_SIZE_CHANNELS_MAX);` `// get CHANNELS-property` `as7352_get_item(DEV_ID, ITEM_ID_CHANNELS, (void *)channels, ITEM_SIZE_CHANNELS_MAX);` **See also** as7352_channels

Enumerator

ITEM_ID_VERSION	Every byte describes one version position: MAJOR MINOR PATCH BUILD. See structure as7352_version.
	
	Properties
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_VERSION (unsigned 4byte)
	• *Parameter range:* 0 - 255, 4 times
	• *Default value:* no default
	• *Direction:* read only
	
	Example
	`struct as7352_version version;`
	`as7352_get_item(DEV_ID, ITEM_ID_VERSION, (void *)&version,`
	`ITEM_SIZE_VERSION);`
	
	See also
	as7352_version

ITEM_ID_SERIAL	Unique chip identifier. See structure as7352_serial.
	
	Properties
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_SERIAL, see as7352_serial
	• *Parameter range:* see definition of as7352_serial
	• *Default value:* no default
	• *Direction:* read only
	
	Example
	`struct as7352_serial serial;`
	`as7352_get_item(DEV_ID, ITEM_ID_SERIAL, (void *)&serial,`
	`ITEM_SIZE_SERIAL);`
	
	See also
	as7352_serial

Enumerator

ITEM_ID_AUTOZERO	This item accesses directly the AS7352 AZ_CONFIG-register 0xD6. The register configures how often the spectral engine offsets are reset (auto zero) to compensate for changes of the device temperature. The typical time auto zero needs to be completed is 15ms.
	Parameter description:
	• 0: Never used
	• 1: Every integration cycle
	• 2: Every 2 integration cycle
	• ...
	• 254: Every 254 cycles
	• 255: Only before first measurement
	
	Properties
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_AUTOZERO (unsigned 8bit)
	• *Parameter range:* 0 - 255
	• *Default value:* 255
	• *Direction:* write/read
	Example
	`uint8_t autozero_set, autozero_get;`
	`// set AUTOZERO-property`
	`autozero_set = 0;`
	`as7352_set_item(DEV_ID, ITEM_ID_AUTOZERO, (void *)&autozero_set, ITEM_SIZE_AUTOZERO);`
	`// get AUTOZERO-property`
	`as7352_get_item(DEV_ID, ITEM_ID_AUTOZERO, (void *)&autozero_get, ITEM_SIZE_AUTOZERO);`
	Attention
	The defintion of this item is valid, if at maximum 6 channels are configured. If more than 6 channels are used, auto zero will be restarted after every channel reconfiguration. That means, that value 1 - 255 has the same effect: Auto zero is used on every measurement.

Enumerator

ITEM_ID_MEAS_COUNT_ALS	Configured the number of measurements, 0 means continuous measurement.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL, MEASUREMENT_TYPE_SPECTRAL_FIFO
	• *Payload size:* ITEM_SIZE_MEAS_COUNT_ALS (unsigned 16bit)
	• *Parameter range:* 0 - 65535
	• *Default value:* 0 (continuous measurement)
	• *Direction:* write/read
	
	Example
	
	`uint16_t meas_count_set, meas_count_get;`
	`// set MEAS_COUNT-property`
	`meas_count_set = 5;`
	`as7352_set_item(DEV_ID, ITEM_ID_MEAS_COUNT_ALS, (void *)&meas_count_set, ITEM_SIZE_MEAS_COUNT_ALS);`
	`// get MEAS_COUNT-property`
	`as7352_get_item(DEV_ID, ITEM_ID_MEAS_COUNT_ALS, (void *)&meas_count_get, ITEM_SIZE_MEAS_COUNT_ALS);`
	
	Note
	
	For measurement mode MEASUREMENT_TYPE_SPECTRAL_FIFO both items ITEM_ID_MEAS_COUNT_ALS and ITEM_ID_MEAS_COUNT_FIFO can be set. Whichever value is reached first will cause the measurment to abort

Enumerator

ITEM_ID_LED_PATTERN	With the help of the structure as7352_led_pattern LEDs can be switched synchronously to the measurement. Inside the structure can be defined how long a specifc LED configuration shall be used.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_LED_PATTERN (size of structure as7352_led_pattern, 10x)
	• *Parameter range:* count: 0 - 255, config: see as7352_led_masks
	• *Default value:* count: 0, config: LED_MASK_OFF
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_pattern led_pattern[AS7352_LED_PATTERN_NUM] = {0, 0};`
	`// configure the LED pattern:`
	`// first two measurements:  LEDs off,`
	`led_pattern[0].config = LED_MASK_OFF;`
	`led_pattern[0].count = 2;`
	`// next three measurements:  internal LED activated`
	`led_pattern[1].config = LED_MASK_INTERN;`
	`led_pattern[1].count = 3;`
	`// next four measurements:  internal LED and external LED_0 are activated`
	`led_pattern[2].config = LED_MASK_EXT_0 | LED_MASK_INTERN;`
	`led_pattern[2].count = 4;`
	`// set LED_PATTERN-property`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_PATTERN, (void *)led_pattern, ITEM_SIZE_LED_PATTERN);`
	`// get LED_PATTERN-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_PATTERN, (void *)led_pattern, ITEM_SIZE_LED_PATTERN);`

ITEM_ID_LED_WAIT_TIME	Set warm up time in microseconds for synchronized LED switching.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_LED_WAIT_TIME (unsigned 32bit)
	• *Parameter range:* 0us - 10s
	• *Default value:* 100000 (100ms)
	• *Direction:* write/read
	
	Example
	
	`uint32_t warm_up_time_set, warm_up_time_get;`
	`// set LED_WAIT_TIME-property`
	`warm_up_time_set = 0;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_WAIT_TIME, (void *)&warm_up_time_set, ITEM_SIZE_LED_WAIT_TIME);`
	`// get LED_WAIT_TIME-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_WAIT_TIME, (void *)&warm_up_time_get, ITEM_SIZE_LED_WAIT_TIME);`

Enumerator

ITEM_ID_INTERRUPT_PIN	Unequal zero means that the interrupt pin will be used, otherwise the time of integration will be calculated internally.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL, MEASUREMENT_TYPE_FIFO
	• *Payload size:* ITEM_SIZE_INTERRUPT_PIN (unsigned 8bit)
	• *Parameter range:* 0 - 1
	• *Default value:* 0
	• *Direction:* write/read
	
	Example
	
	`uint8_t interrupt_pin_set, interrupt_pin_get;`
	`// set INTERRUPT_PIN-property`
	`interrupt_pin_set = 1;`
	`as7352_set_item(DEV_ID, ITEM_ID_INTERRUPT_PIN, (void *)&interrupt_pin_set, ITEM_SIZE_INTERRUPT_PIN);`
	`// get INTERRUPT_PIN-property`
	`as7352_get_item(DEV_ID, ITEM_ID_INTERRUPT_PIN, (void *)&interrupt_pin_get, ITEM_SIZE_INTERRUPT_PIN);`

ITEM_ID_LED_INTERN	Configures the internal LED with help of structure as7352_led_config.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_LED_INTERN (twice unsigned 16bit)
	• *Parameter range:* see as7352_led_config
	• *Default value:* enable: 0, brightness: 100
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_config config_set, config_get;`
	`// set LED_INTERN-property`
	`config_set.enable = 1;`
	`config_set.brightness = 1000;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_INTERN, (void *)&config_set, ITEM_SIZE_LED_INTERN);`
	`// get LED_INTERN-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_INTERN, (void *)&config_get, ITEM_SIZE_LED_INTERN);`

Enumerator

ITEM_ID_LED_EXT_0	Configures the external LED 0 with help of structure as7352_led_config.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_LED_EXT_0 (twice unsigned 16bit)
	• *Parameter range:* see as7352_led_config
	• *Default value:* enable: 0, brightness: 100
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_config config_set, config_get;`
	`// set LED_EXT_0-property`
	`config_set.enable = 1;`
	`config_set.brightness = 1000;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_EXT_0, (void *)&config_set, ITEM_SIZE_LED_EXT_0);`
	`// get LED_EXT_0-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_EXT_0, (void *)&config_get, ITEM_SIZE_LED_EXT_0);`

ITEM_ID_LED_EXT_1	Configures the external LED 1 with help of structure as7352_led_config.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_LED_EXT_1 (twice unsigned 16bit)
	• *Parameter range:* see as7352_led_config
	• *Default value:* enable: 0, brightness: 100
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_config config_set, config_get;`
	`// set LED_EXT_1-property`
	`config_set.enable = 1;`
	`config_set.brightness = 1000;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_EXT_1, (void *)&config_set, ITEM_SIZE_LED_EXT_1);`
	`// get LED_EXT_1-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_EXT_1, (void *)&config_get, ITEM_SIZE_LED_EXT_1);`

Enumerator

ITEM_ID_LED_EXT_2	Configures the external LED 2 with help of structure as7352_led_config.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_LED_EXT_2 (twice unsigned 16bit)
	• *Parameter range:* see as7352_led_config
	• *Default value:* enable: 0, brightness: 100
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_config config_set, config_get;`
	`// set LED_EXT_2-property`
	`config_set.enable = 1;`
	`config_set.brightness = 1000;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_EXT_2, (void *)&config_set,`
	`ITEM_SIZE_LED_EXT_2);`
	`// get LED_EXT_2-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_EXT_2, (void *)&config_get,`
	`ITEM_SIZE_LED_EXT_2);`

ITEM_ID_LED_EXT_3	Configures the external LED 3 with help of structure as7352_led_config.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_LED_EXT_3 (twice unsigned 16bit)
	• *Parameter range:* see as7352_led_config
	• *Default value:* enable: 0, brightness: 100
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_config config_set, config_get;`
	`// set LED_EXT_3-property`
	`config_set.enable = 1;`
	`config_set.brightness = 1000;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_EXT_3, (void *)&config_set,`
	`ITEM_SIZE_LED_EXT_3);`
	`// get LED_EXT_3-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_EXT_3, (void *)&config_get,`
	`ITEM_SIZE_LED_EXT_3);`

Enumerator

ITEM_ID_LED_EXT_4	Configures the external LED 4 with help of structure as7352_led_config.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_LED_EXT_4 (twice unsigned 16bit)
	• *Parameter range:* see as7352_led_config
	• *Default value:* enable: 0, brightness: 100
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_config config_set, config_get;`
	`// set LED_EXT_4-property`
	`config_set.enable = 1;`
	`config_set.brightness = 1000;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_EXT_4, (void *)&config_set, ITEM_SIZE_LED_EXT_4);`
	`// get LED_EXT_4-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_EXT_4, (void *)&config_get, ITEM_SIZE_LED_EXT_4);`

ITEM_ID_LED_EXT_5	Configures the external LED 5 with help of structure as7352_led_config.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_LED_EXT_5 (twice unsigned 16bit)
	• *Parameter range:* see as7352_led_config
	• *Default value:* enable: 0, brightness: 100
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_config config_set, config_get;`
	`// set LED_EXT_5-property`
	`config_set.enable = 1;`
	`config_set.brightness = 1000;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_EXT_5, (void *)&config_set, ITEM_SIZE_LED_EXT_5);`
	`// get LED_EXT_5-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_EXT_5, (void *)&config_get, ITEM_SIZE_LED_EXT_5);`

Enumerator

ITEM_ID_OUTPUT	The output is configured as an open collector. On default the output is HI-Z and is disconnected. Unequal zero means that the output pin drives low.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_OUTPUT (unsigned 8bit)
	• *Parameter range:* 0 (HI-Z) - 1 (GND)
	• *Default value:* 0 (HI-Z)
	• *Direction:* write/read
	
	Example
	
	`uint8_t output_set, output_get;`
	`// set OUTPUT-property`
	`output_set = 1`
	`as7352_set_item(DEV_ID, ITEM_ID_OUTPUT, (void *)&output_set,`
	`ITEM_SIZE_OUTPUT);`
	`// get OUTPUT-property`
	`as7352_get_item(DEV_ID, ITEM_ID_OUTPUT, (void *)&output_get,`
	`ITEM_SIZE_OUTPUT);`

ITEM_ID_TEMP_EXT_0	Get the temperature of an external sensor 0 in millidegree.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_TEMP_EXT_0 (signed 32bit)
	• *Parameter range:* depends on OSAL implementation (typical -40000 ... 125000)
	• *Default value:* -
	• *Direction:* read
	
	Example
	
	`int32_t millidegree_get;`
	`// get TEMP_EXT_0-property`
	`as7352_get_item(DEV_ID, ITEM_ID_TEMP_EXT_0, (void`
	`*)&millidegree_get, ITEM_SIZE_TEMP_EXT_0);`

Enumerator

ITEM_ID_TEMP_EXT_1	Get the temperature of an external sensor 1 in millidegree.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_TEMP_EXT_1 (signed 32bit)
	• *Parameter range:* depends on OSAL implementation (typical -40000 ... 125000)
	• *Default value:* -
	• *Direction:* read
	
	Example
	
	`int32_t millidegree_get;`
	`// get TEMP_EXT_1-property`
	`as7352_get_item(DEV_ID, ITEM_ID_TEMP_EXT_1, (void *)&millidegree_get, ITEM_SIZE_TEMP_EXT_1);`

ITEM_ID_TEMP_EXT_2	Get the temperature of an external sensor 2 in millidegree.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_TEMP_EXT_2 (signed 32bit)
	• *Parameter range:* depends on OSAL implementation (typical -40000 ... 125000)
	• *Default value:* -
	• *Direction:* read
	
	Example
	
	`int32_t millidegree_get;`
	`// get TEMP_EXT_2-property`
	`as7352_get_item(DEV_ID, ITEM_ID_TEMP_EXT_2, (void *)&millidegree_get, ITEM_SIZE_TEMP_EXT_2);`

Enumerator

ITEM_ID_TEMP_EXT_3	Get the temperature of an external sensor 3 in millidegree.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_TEMP_EXT_3 (signed 32bit)
	• *Parameter range:* depends on OSAL implementation (typical -40000 ... 125000)
	• *Default value:* -
	• *Direction:* read
	
	Example
	
	`int32_t millidegree_get;`
	`// get TEMP_EXT_3-property`
	`as7352_get_item(DEV_ID, ITEM_ID_TEMP_EXT_3, (void *)&millidegree_get, ITEM_SIZE_TEMP_EXT_3);`

ITEM_ID_TEMP_EXT_4	Get the temperature of an external sensor 4 in millidegree.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_TEMP_EXT_4 (signed 32bit)
	• *Parameter range:* depends on OSAL implementation (typical -40000 ... 125000)
	• *Default value:* -
	• *Direction:* read
	
	Example
	
	`int32_t millidegree_get;`
	`// get TEMP_EXT_4-property`
	`as7352_get_item(DEV_ID, ITEM_ID_TEMP_EXT_4, (void *)&millidegree_get, ITEM_SIZE_TEMP_EXT_4);`

Enumerator

ITEM_ID_TEMP_EXT_5	Get the temperature of an external sensor 5 in millidegree.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_TEMP_EXT_5 (signed 32bit)
	• *Parameter range:* depends on OSAL implementation (typical -40000 ... 125000)
	• *Default value:* -
	• *Direction:* read
	
	Example
	
	`int32_t millidegree_get;`
	`// get TEMP_EXT_5-property`
	`as7352_get_item(DEV_ID, ITEM_ID_TEMP_EXT_5, (void *)&millidegree_get, ITEM_SIZE_TEMP_EXT_5);`

ITEM_ID_MEASURE_ITEMS	Item describes additional measurement data. The list can be fullfilled with any item which shall be readout synchronized to the measurement. Those item content will be transferred with the callback function.
	
	Note
	
	The maximum supported buffer size in callback function is AS7352_MAX_ITEM_BUFFER_SIZE.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL, MEASUREMENT_TYPE_FIFO
	• *Payload size:* ITEM_SIZE_MEASURE_ITEMS (10x unsigned 8bit)
	• *Parameter range:* 0 .. ITEM_ID_MAX - 1
	• *Default value:* ITEM_ID_RESERVED (10 times)
	• *Direction:* write/read
	
	Example
	
	`uint8_t measure_items[ITEM_SIZE_MEASURE_ITEMS] = {ITEM_ID_RESERVED};`
	`// set MEASURE_ITEMS-property: added items for temperature sensor`
	`// and gain to cyclic measurement data`
	`measure_items[0] = ITEM_ID_TEMP_EXT_0;`
	`measure_items[1] = ITEM_ID_AGAIN;`
	`as7352_set_item(DEV_ID, ITEM_ID_MEASURE_ITEMS, (void *)&measure_items, ITEM_SIZE_MEASURE_ITEMS);`
	`// get MEASURE_ITEMS-property`
	`as7352_get_item(DEV_ID, ITEM_ID_MEASURE_ITEMS (void *)&measure_items, ITEM_SIZE_MEASURE_ITEMS);`

Enumerator

ITEM_ID_FGAIN	Configures the GAIN for fast FIFO measurement mode. Internally, the register FD_TIME_2 is used.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_FIFO
	• *Payload size:* ITEM_SIZE_FGAIN (unsigned 8bit)
	• *Parameter range:* see as7352_gain
	• *Default value:* GAIN_16X
	• *Direction:* write/read
	
	Example
	
	`uint8_t fgain_set, fgain_get;`
	`// set FGAIN-property`
	`fgain_set = GAIN_256X;`
	`as7352_set_item(DEV_ID, ITEM_ID_FGAIN, (void *)&fgain_set,`
	`ITEM_SIZE_FGAIN);`
	`// get FGAIN-property`
	`as7352_get_item(DEV_ID, ITEM_ID_FGAIN, (void *)&fgain_get,`
	`ITEM_SIZE_FGAIN);`

Enumerator

ITEM_ID_FTIME	Configures the integration time for fast FIFO measurement mode: This parameter sets the I2C registers FD_TIME_1 and FD_TIME_2 directly.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_FIFO
	• *Payload size:* ITEM_SIZE_FTIME (unsigned 16bit)
	• *Parameter range:* 0 - 0x07FF
	• *Default value:* 359
	• *Direction:* write/read
	
	Note
	
	If FTIME is smaller or equal 255, the resulting ADC values also cannot be greater than 255. Therefore, the 8-Bit FIFO mode is automatically used. In this mode, the FIFO only stores 8-Bit values, which can be read out in half the time. Other than speed, the 8-Bit mode has no effect on data aquisition. The data will still be saved in the 16-Bit data buffer to remain compatible with the rest of the chip library.
	
	Example
	
	`uint16_t ftime_set, ftime_get;`
	`// set FTIME-property`
	`ftime_set = 179;`
	`as7352_set_item(DEV_ID, ITEM_ID_FTIME, (void *)&ftime_set,`
	`ITEM_SIZE_FTIME);`
	`// get FTIME-property`
	`as7352_get_item(DEV_ID, ITEM_ID_FTIME, (void *)&ftime_get,`
	`ITEM_SIZE_FTIME);`
	
	See also
	
	ITEM_ID_FTIME_US

Enumerator

ITEM_ID_FTIME_US	Configures the integration time for fast FIFO measurement mode in microseconds.
	Following formula describes the relationship:
	$t_{int} = (FD\_TIME + 1) * (2000/720)us$
	The maximum possible ADC fullscale range has a width of 16bit, but is limited by the integration time:
	$ADC_{fullscale} = FD\_TIME + 1$
	Properties
	• *Measurement mode:* MEASUREMENT_TYPE_FIFO
	• *Payload size:* ITEM_SIZE_FTIME_US (unsigned 32bit)
	• *Parameter range:* 3us - 5689us
	• *Default value:* 1000us
	• *Direction:* write/read
	Example
	`uint32_t ftime_us_set, ftime_us_get;`
	`// set FTIME_US-property`
	`ftime_set = 500; // set to 500us == 2kHz`
	`as7352_set_item(DEV_ID, ITEM_ID_FTIME_US, (void *)&ftime_us_set, ITEM_SIZE_FTIME_US);`
	`// get FTIME_US-property`
	`as7352_get_item(DEV_ID, ITEM_ID_FTIME_US, (void *)&ftime_us_get, ITEM_SIZE_FTIME_US);`

ITEM_ID_TIMESTAMP	Reads a microseconds timestamp. It can be used to get accurate time difference between to measurements.
	Properties
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_TIMESTAMP (unsigned 64bit)
	• *Parameter range:* full 64bit range
	• *Default value:* -
	• *Direction:* read only
	Example
	`uint64_t timestamp_get;`
	`// get TIMESTAMP-property`
	`as7352_get_item(DEV_ID, ITEM_ID_TIMESTAMP, (void *)&timestamp_get, ITEM_SIZE_TIMESTAMP);`

Enumerator

ITEM_ID_AUTO_GAIN_RANGE_ALS	Enables or disables the automatic gain control for spectral measurements. If lower and upper limit are different and the lower value is lower than the upper one, auto gain is enabled. In this case the gain will be changed automatically. The algorithm tries to map the highest ADC-value to 80% of the maximum. The configured default gain of item ITEM_ID_AGAIN will be used as start value. If this value is out of the configured range, then the lower limit will be used instead. Dependent on the current measurement mode, the item sets ITEM_ID_AGAIN or will be set internally.
	Note
	It's usefull to configure the item ITEM_ID_MEASURE_ITEMS with ITEM_ID_AGAIN to normalize the measured data in the application software.
	Properties
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL, MEASUREMENT_TYPE_SPECTRAL_FIFO
	• *Payload size:* ITEM_SIZE_AUTO_GAIN_RANGE_ALS (see as7352_auto_gain)
	• *Parameter range:* see as7352_gain
	• *Default value:* GAIN_0_5X, GAIN_0_5X
	• *Direction:* write/read
	Example
	`struct as7352_auto_gain auto_gain;`
	`// Enable auto gain over the whole range.`
	`auto_gain.lower_limit = GAIN_0_5X;`
	`auto_gain.upper_limit = GAIN_5120X,`
	`as7352_set_item(DEV_ID, ITEM_ID_AUTO_GAIN_RANGE_ALS, (void *)&auto_gain, ITEM_SIZE_AUTO_GAIN_RANGE_ALS);`

Enumerator

ITEM_ID_AUTO_GAIN_RANGE_FIFO	Enables or disables the automatic gain control for FiFo measurements. If lower and upper limit are different and the lower value is lower than the upper one, auto gain is enabled. In this case the gain will be changed automatically. The algorithm tries to map the highest ADC-value to 80% of the maximum. The configured default gain of item ITEM_ID_FGAIN will be used as start value. If this value is out of the configured range, then the lower limit will be used instead. Dependent on the current measurement mode, the item sets ITEM_ID_AGAIN or ITEM_ID_FGAIN will be set internally.
	Note
	It's useful to configure the item ITEM_ID_MEASURE_ITEMS with ITEM_ID_FGAIN to normalize the measured data in the application software.
	Properties
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL_FIFO, MEASUREMENT_TYPE_FIFO
	• *Payload size:* ITEM_SIZE_AUTO_GAIN_RANGE_FIFO (see as7352_auto_gain)
	• *Parameter range:* see as7352_gain
	• *Default value:* GAIN_0_5X, GAIN_0_5X
	• *Direction:* write/read
	Example
	`struct as7352_auto_gain auto_gain;`
	`// Enable auto gain over the whole range.`
	`auto_gain.lower_limit = GAIN_0_5X;`
	`auto_gain.upper_limit = GAIN_5120X,`
	`as7352_set_item(DEV_ID, ITEM_ID_AUTO_GAIN_RANGE_ALS, (void *)&auto_gain, ITEM_SIZE_AUTO_GAIN_RANGE_FIFO);`

Enumerator

ITEM_ID_GAIN_FACTORS	The sensor AS7352 has a gain error over the whole area. If the gain will be increased by one, the measured value is not exact the double. To compensate this behavior, the measured values will be multiplied with a correction factor. The default values are used from the datasheet but can be changed.
	The given factors will be divided by 10000 internally to get the right factor with four decimal places.
	Note
	The feature can be disabled by setting all factors to 10000.
	Properties
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_GAIN_FACTORS (15x unsigned short)
	• *Parameter range:* 1 - 20000
	• *Default value:* 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000 (gain GAIN_0_5X - GAIN_5120X)
	• *Direction:* write/read
	Note
	TODO: Gain factors are set to 100%. After validation these should be updated
	Example
	`uint16_t gain_factors[AS7352_GAIN_FACTOR_NUM];`
	`// Read gain factors`
	`as7352_get_item(DEV_ID, ITEM_ID_GAIN_FACTORS, (void *)gain_factors, ITEM_SIZE_GAIN_FACTORS);`

Enumerator

ITEM_ID_REG_READ	Read a register. This item implements a read register access as a two step process where the register address needs to written to the item first. The subsequent read of this item will return the register value. For register addresses < 0x80, the register bank switch is performed automatically, if necessary. After a successful read operation, the address pointer will be incremented by 1
	Note
	On initialization, the address pointer is at 0x00
	Properties
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_REG_READ (unsigned 8bit)
	• *Parameter range:* 0 - 255
	• *Default value:* 0
	• *Direction:* read / write
	Example
	`uint8_t reg_addr = AS7352_REGADDR_STATUS uint8_t value;`
	`// set address pointer to AS7352_REGADDR_STATUS`
	`as7352_set_item(DEV_ID, ITEM_ID_REG_READ, (void *)&reg_addr, sizeof(uint8_t));`
	`// read register AS7352_REGADDR_STATUS`
	`as7352_get_item(DEV_ID, ITEM_ID_REG_READ, (void *)&value, ITEM_SIZE_REG_READ);`

ITEM_ID_SYNC_MODE	Configures the sync mode of the sensor. When sync mode is enabled, the sensor triggers a measurement either on a rising or falling edge of the signal connected to the GPIO pin.
	Properties
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_SYNC_MODE (unsigned 8bit)
	• *Parameter range:* see as7352_sync_mode
	• *Default value:* SYNC_MODE_DISABLED
	• *Direction:* read / write
	Example
	`uint8_t sync_mode = SYNC_MODE_DISABLED;`
	`as7352_set_item(DEV_ID, ITEM_ID_SYNC_MODE, (void *)&sync_mode, ITEM_SIZE_SYNC_MODE);`
	`as7352_get_item(DEV_ID, ITEM_ID_SYNC_MODE, (void *)&sync_mode, ITEM_SIZE_SYNC_MODE);`

Enumerator

ITEM_ID_FIFO_AGC_BLOCKSIZE	Configures the blocksize for fifo measurements, after every block a AGC cycle will be executed. So the application can ensure that it will receive at least one consistent block between 2 AGC cycles to calculate a FFT from it. So, a blocksize should be >= FFT window size. A blocksize of 0 will cause an AGC cycle at the beginning of measurement and will then run infinetely without any further AGC.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_FIFO_AGC_BLOCKSIZE (unsigned 8bit)
	• *Parameter range:* full 16bit range
	• *Default value:* 0
	• *Direction:* read / write
	
	Example
	
	`uint16_t block_size = 1024;`
	`as7352_set_item(DEV_ID, ITEM_ID_FIFO_AGC_BLOCKSIZE, (void *)&block_size, ITEM_SIZE_FIFO_AGC_BLOCKSIZE);`
	`as7352_get_item(DEV_ID, ITEM_ID_FIFO_AGC_BLOCKSIZE, (void *)&block_size, ITEM_SIZE_FIFO_AGC_BLOCKSIZE);`

ITEM_ID_FD_COEFF	Configure the number of datasets for the detection of the flicker frequency using onchip Goertzel algorithm.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_GOERTZEL
	• *Payload size:* ITEM_SIZE_FD_COEFF (unsigned 8bit)
	• *Parameter range:* 0-7
	• *Default value:* 4
	• *Direction:* read / write
	
	Example
	
	`uint16_t fd_coeff = 3;`
	`as7352_set_item(DEV_ID, ITEM_ID_FD_COEFF, (void *)&fd_coeff, ITEM_SIZE_FD_COEFF);`
	`as7352_get_item(DEV_ID, ITEM_ID_FD_COEFF, (void *)&fd_coeff, ITEM_SIZE_FD_COEFF);`

Enumerator

ITEM_ID_FD_DCR	Configure the window size of the DC removal for the detection of the flicker frequency using onchip Goertzel algorithm.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_GOERTZEL
	• *Payload size:* ITEM_SIZE_FD_DCR (unsigned 8bit)
	• *Parameter range:* 0-7
	• *Default value:* 3
	• *Direction:* read / write
	
	Example
	
	`uint16_t fd_dcr = 3;`
	`as7352_set_item(DEV_ID, ITEM_ID_FD_DCR, (void *)&fd_dcr,`
	`ITEM_SIZE_FD_DCR);`
	`as7352_get_item(DEV_ID, ITEM_ID_FD_DCR, (void *)&fd_dcr,`
	`ITEM_SIZE_FD_DCR);`

ITEM_ID_FD_PERS	Configure the number of consecutive flicker frequency detections, so that the detection is accepted to be valid at onchip Goertzel algorithm.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_GOERTZEL
	• *Payload size:* ITEM_SIZE_FD_PERS (unsigned 8bit)
	• *Parameter range:* 0-7
	• *Default value:* 2
	• *Direction:* read / write
	
	Example
	
	`uint16_t fd_pers = 3;`
	`as7352_set_item(DEV_ID, ITEM_ID_FD_PERS, (void *)&fd_pers,`
	`ITEM_SIZE_FD_PERS);`
	`as7352_get_item(DEV_ID, ITEM_ID_FD_PERS, (void *)&fd_pers,`
	`ITEM_SIZE_FD_PERS);`

Enumerator

ITEM_ID_FD_AGC_DISABLE	Disable (1) or enable (0) the automatic gain control for detection of flicker frequencies at onchip Goertzel algorithm.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_GOERTZEL
	• *Payload size:* ITEM_SIZE_FD_AGC_DISABLE (unsigned 8bit)
	• *Parameter range:* 0-1
	• *Default value:* 0
	• *Direction:* read / write
	
	Example
	
	`uint16_t fd_agc_disable = 1;`
	`as7352_set_item(DEV_ID, ITEM_ID_FD_AGC_DISABLE, (void *)&fd_agc_disable, ITEM_SIZE_FD_AGC_DISABLE);`
	`as7352_get_item(DEV_ID, ITEM_ID_FD_AGC_DISABLE, (void *)&fd_agc_disable, ITEM_SIZE_FD_AGC_DISABLE);`

ITEM_ID_FD_MODCLK	Configure the clock speeds for detection of the flicker frequency using onchip Goertzel algorithm.
	
	• Bits 0-1: Modclk speed dur-ing flicker integration.
	• Bits 2-3: Modclk speed dur-ing flicker residual measurements
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_GOERTZEL
	• *Payload size:* ITEM_SIZE_FD_MODCLK (unsigned 8bit)
	• *Parameter range:* Bits[0:1] 0-3 ; Bits[2:3] 0-3
	• *Default value:* Bits[0:1] 0 ; Bits[2:3] 0
	• *Direction:* read / write
	
	Example
	
	`uint16_t fd_mod_clk = 3;`
	`as7352_set_item(DEV_ID, ITEM_ID_FD_MODCLK, (void *)&fd_mod_clk, ITEM_SIZE_FD_MODCLK);`
	`as7352_get_item(DEV_ID, ITEM_ID_FD_MODCLK, (void *)&fd_mod_clk, ITEM_SIZE_FD_MODCLK);`

Enumerator

ITEM_ID_FD_SAMPLES	Configure the number of samples used for detection of the flicker frequency using onchip Goertzel algorithm.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_GOERTZEL
	• *Payload size:* ITEM_SIZE_FD_SAMPLES (unsigned 8bit)
	• *Parameter range:* 0-7
	• *Default value:* 4
	• *Direction:* read / write
	
	Example
	
	`uint16_t fd_samples = 6;`
	`as7352_set_item(DEV_ID, ITEM_ID_FD_SAMPLES, (void *)&fd_samples, ITEM_SIZE_FD_SAMPLES);`
	`as7352_get_item(DEV_ID, ITEM_ID_FD_SAMPLES, (void *)&fd_samples, ITEM_SIZE_FD_SAMPLES);`

ITEM_ID_FD_COMPARE_VALUE	Configure the compare limit for the detection of the flick-er frequency using onchip Goertzel algorithm.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_GOERTZEL
	• *Payload size:* ITEM_SIZE_FD_COMPARE_VALUE (unsigned 8bit)
	• *Parameter range:* 0-7
	• *Default value:* 1
	• *Direction:* read / write
	
	Example
	
	`uint16_t fd_compare_value = 3;`
	`as7352_set_item(DEV_ID, ITEM_ID_FD_COMPARE_VALUE, (void *)&fd_compare_value, ITEM_SIZE_FD_COMPARE_VALUE);`
	`as7352_get_item(DEV_ID, ITEM_ID_FD_COMPARE_VALUE, (void *)&fd_compare_value, ITEM_SIZE_FD_COMPARE_VALUE);`

Enumerator

ITEM_ID_MEAS_COUNT_FIFO	Configure the number of measurements in a buffered measurement, 0 means continuous measurement.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_FIFO, MEASUREMENT_TYPE_SPECTRAL_FIFO
	• *Payload size:* ITEM_SIZE_MEAS_COUNT_FIFO (unsigned 32bit)
	• *Parameter range:* 0 - 0xFFFFFFFF
	• *Default value:* 0 (continuous measurement)
	• *Direction:* write/read
	
	Example
	
	`uint32_t meas_count_set, meas_count_get;`
	`// set MEAS_COUNT-property`
	`meas_count_set = 20000;`
	`as7352_set_item(DEV_ID, ITEM_ID_MEAS_COUNT_FIFO, (void *)&meas_count_set, ITEM_SIZE_MEAS_COUNT_FIFO);`
	`// get MEAS_COUNT-property`
	`as7352_get_item(DEV_ID, ITEM_ID_MEAS_COUNT_FIFO, (void *)&meas_count_get, ITEM_SIZE_MEAS_COUNT_FIFO);`
	
	Note
	
	For measurement mode MEASUREMENT_TYPE_SPECTRAL_FIFO both items ITEM_ID_MEAS_COUNT_ALS and ITEM_ID_MEAS_COUNT_FIFO can be set. Whichever value is reached first will cause the measurement to abort

ITEM_ID_MAX	Internal definition of maximum supported items.

4.3.4.8 as7352_item_sizes `enum as7352_item_sizes`

List, which describes the supported length of every item.

Enumerator

ITEM_SIZE_RESERVED	size in bytes of item ITEM_ID_RESERVED
ITEM_SIZE_ASTEP	size in bytes of item ITEM_ID_ASTEP
ITEM_SIZE_ATIME	size in bytes of item ITEM_ID_ATIME
ITEM_SIZE_ITIME	size in bytes of item ITEM_ID_ITIME
ITEM_SIZE_AGAIN	size in bytes of item ITEM_ID_AGAIN
ITEM_SIZE_MEAS_TYPE	size in bytes of item ITEM_ID_MEAS_TYPE
ITEM_SIZE_BREAK	size in bytes of item ITEM_ID_BREAK
ITEM_SIZE_CHANNELS_MAX	size in bytes of item ITEM_ID_CHANNELS

Enumerator

ITEM_SIZE_VERSION	size in bytes of item ITEM_ID_VERSION
ITEM_SIZE_SERIAL	size in bytes of item ITEM_ID_SERIAL
ITEM_SIZE_SATURATION	size in bytes of item ITEM_ID_SATURATION
ITEM_SIZE_AUTOZERO	size in bytes of item ITEM_ID_AUTOZERO
ITEM_SIZE_MEAS_COUNT_ALS	size in bytes of item ITEM_ID_MEAS_COUNT_ALS
ITEM_SIZE_MEAS_COUNT_FIFO	size in bytes of item ITEM_ID_MEAS_COUNT_FIFO
ITEM_SIZE_LED_PATTERN	size in bytes of item ITEM_ID_LED_PATTERN
ITEM_SIZE_LED_WAIT_TIME	size in bytes of item ITEM_ID_LED_WAIT_TIME
ITEM_SIZE_INTERRUPT_PIN	size in bytes of item ITEM_ID_INTERRUPT_PIN
ITEM_SIZE_LED_INTERN	size in bytes of item ITEM_ID_LED_INTERN
ITEM_SIZE_LED_EXT_0	size in bytes of item ITEM_ID_LED_EXT_0
ITEM_SIZE_LED_EXT_1	size in bytes of item ITEM_ID_LED_EXT_1
ITEM_SIZE_LED_EXT_2	size in bytes of item ITEM_ID_LED_EXT_2
ITEM_SIZE_LED_EXT_3	size in bytes of item ITEM_ID_LED_EXT_3
ITEM_SIZE_LED_EXT_4	size in bytes of item ITEM_ID_LED_EXT_4
ITEM_SIZE_LED_EXT_5	size in bytes of item ITEM_ID_LED_EXT_5
ITEM_SIZE_OUTPUT	size in bytes of item ITEM_ID_OUTPUT
ITEM_SIZE_TEMP_EXT_0	size in bytes of item ITEM_ID_TEMP_EXT_0
ITEM_SIZE_TEMP_EXT_1	size in bytes of item ITEM_ID_TEMP_EXT_1
ITEM_SIZE_TEMP_EXT_2	size in bytes of item ITEM_ID_TEMP_EXT_2
ITEM_SIZE_TEMP_EXT_3	size in bytes of item ITEM_ID_TEMP_EXT_3
ITEM_SIZE_TEMP_EXT_4	size in bytes of item ITEM_ID_TEMP_EXT_4
ITEM_SIZE_TEMP_EXT_5	size in bytes of item ITEM_ID_TEMP_EXT_5
ITEM_SIZE_MEASURE_ITEMS	size in bytes of item ITEM_ID_MEASURE_ITEMS
ITEM_SIZE_FGAIN	size in bytes of item ITEM_ID_FGAIN
ITEM_SIZE_FTIME	size in bytes of item ITEM_ID_FTIME
ITEM_SIZE_FTIME_US	size in bytes of item ITEM_ID_FTIME_US
ITEM_SIZE_TIMESTAMP	size in bytes of item ITEM_ID_TIMESTAMP
ITEM_SIZE_AUTO_GAIN_RANGE_ALS	size in bytes of item ITEM_ID_AUTO_GAIN_ALS
ITEM_SIZE_AUTO_GAIN_RANGE_FIFO	size in bytes of item ITEM_ID_AUTO_GAIN_FIFO
ITEM_SIZE_GAIN_FACTORS	size in bytes of item ITEM_ID_GAIN_FACTOR
ITEM_SIZE_REG_READ	size in bytes of item ITEM_ID_REG_READ
ITEM_SIZE_SYNC_MODE	size in bytes of item ITEM_ID_SYNC_MODE
ITEM_SIZE_FIFO_AGC_BLOCKSIZE	size in bytes of item ITEM_ID_FIFO_AGC_BLOCKSIZE
ITEM_SIZE_FD_COEFF	size in bytes of item ITEM_ID_FD_COEFF
ITEM_SIZE_FD_DCR	size in bytes of item ITEM_ID_FD_DCR
ITEM_SIZE_FD_PERS	size in bytes of item ITEM_ID_FD_PERS
ITEM_SIZE_FD_AGC_DISABLE	size in bytes of item ITEM_ID_FD_AGC_DISABLE
ITEM_SIZE_FD_MODCLK	size in bytes of item ITEM_ID_FD_MODCLK
ITEM_SIZE_FD_SAMPLES	size in bytes of item ITEM_ID_FD_SAMPLES
ITEM_SIZE_FD_COMPARE_VALUE	size in bytes of item ITEM_ID_FD_COMPARE_VALUE

4.3.4.9 as7352_states `enum as7352_states`

This enumeration describes the state of the measurement engine

Enumerator

STATE_CONFIG	No measurement is running.
STATE_MEASURE	Measurement is active.

4.4 AS7352 Chip Library Functions

This is the chip library for ams spectral sensor AS7352.

Functions

- err_code_t CHIPLIB_DECLDIR as7352_initialize (const uint8_t device, const as7352_callback_t p_callback, const void *p_cb_param, const char *p_interface_descr)

  Initializes the library and the device.
- err_code_t CHIPLIB_DECLDIR as7352_shutdown (const uint8_t device)

  Stops all internal actions and power down the device.
- err_code_t CHIPLIB_DECLDIR as7352_set_item (const uint8_t device, const enum as7352_item_ids id, void *p_data, const uint8_t size)

  Set an item.
- err_code_t CHIPLIB_DECLDIR as7352_get_item (const uint8_t device, const enum as7352_item_ids id, void *p_data, const uint8_t size)

  Get an item.
- err_code_t CHIPLIB_DECLDIR as7352_set_configuration (const uint8_t device, uint8_t *p_data, const uint32_t size)

  Configuration of multiple sensor or library items.
- err_code_t CHIPLIB_DECLDIR as7352_get_configuration (const uint8_t device, uint8_t *p_data, uint32_t *p_size)

  Request of all supported sensor and library items.
- err_code_t CHIPLIB_DECLDIR as7352_start_measurement (const uint8_t device)

  Starts a measurement.
- err_code_t CHIPLIB_DECLDIR as7352_execute_state_machine (const uint8_t device, enum as7352_states *p_state)

  Executes a measurement step.
- err_code_t CHIPLIB_DECLDIR as7352_abort_measurement (const uint8_t device)

  Abort a measurement.

4.4.1 Detailed Description

This is the chip library for ams spectral sensor AS7352.

4.4.2 Function Documentation

4.4.2.1 as7352_initialize()

```
err_code_t CHIPLIB_DECLDIR as7352_initialize (
        const uint8_t device,
        const as7352_callback_t p_callback,
        const void * p_cb_param,
        const char * p_interface_descr )
```

Initializes the library and the device.

Following tasks will be done here:

- Initialize hardware abstraction layer.
- Set default configuration values.

Note

This function must be called at first, otherwise all other functions return with error code.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes)
in	*p_callback*	Pointer to the callback function, see as7352_callback_t
in	*p_cb_param*	Optional pointer to an application parameter, which will be transmitted with every callback.
in	*p_interface_descr*	Chiplib forwards this interface description to ::spectral_osal_initialize.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid.
ERR_IDENTIFICATION	The specified sensor was not found.
ERR_DATA_TRANSFER	Communication error to sensor.

4.4.2.2 as7352_shutdown()

```
err_code_t CHIPLIB_DECLDIR as7352_shutdown (
        const uint8_t device )
```

Stops all internal actions and power down the device.

Following tasks will be done here:

- Stops measurement, if running.
- Power down the sensor device.
- Shutdown the hardware abstraction layer.
- Block calling of all other functions, but initialize.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes).

Return values

ERR_SUCCESS	Function returns without error.
ERR_DATA_TRANSFER	Communication error to sensor.

4.4.2.3 as7352_set_item()

```
err_code_t CHIPLIB_DECLDIR as7352_set_item (
    const uint8_t device,
    const enum as7352_item_ids id,
    void * p_data,
    const uint8_t size )
```

Set an item.

This function configures the chip library or the sensor directly.

Following tasks will be done here:

- Searches for the corresponding internal configuration function of this item-ID.
- Calls the internal set-function if available.

Note

This function can only called if ChipLib is in state configuration.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes).
in	*id*	Identification number of an item, see as7352_item_ids.
in	*p_data*	Pointer to the data of the item.
in	*size*	Sets the size in byte of data, see as7352_item_sizes.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	Access to the library is blocked, call as7352_initialize at first.
ERR_DATA_TRANSFER	Communication error to sensor.
ERR_NOT_SUPPORTED	e.g. Configuration of item ITEM_ID_LED_EXT_0, but LED is not implemented. Used item-ID is not writable.

4.4.2.4 as7352_get_item()

```
err_code_t CHIPLIB_DECLDIR as7352_get_item (
        const uint8_t device,
        const enum as7352_item_ids id,
        void * p_data,
        const uint8_t size )
```

Get an item.

Reads the actual settings of sensor or library items.

Following tasks will be done here:

- Searches for the corresponding internal request function of this item-ID.
- Calls the internal request-function if available.

Note

This function can be called in state configuration or measurement.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes).
in	*id*	Identification number of an item, see as7352_item_ids.
out	*p_data*	Pointer, where the data of the item can be saved.
in	*size*	Size in byte of data, see as7352_item_sizes.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	Access to the library is blocked, call as7352_initialize at first.
ERR_DATA_TRANSFER	Communication error to sensor
ERR_NOT_SUPPORTED	e.g. Readout of item ITEM_ID_TEMP_EXT_2, but function is not implemented.

4.4.2.5 as7352_set_configuration()

```
err_code_t CHIPLIB_DECLDIR as7352_set_configuration (
        const uint8_t device,
        uint8_t * p_data,
        const uint32_t size )
```

Configuration of multiple sensor or library items.

Following tasks will be done here:

- Searches for the corresponding internal configuration function of the given item-IDs.
- Calls the internal set-function if available.

Byte sequence of data [S0][I0][D0_0 - D0_N][S1][I1][D1_0 - D1_N] ... [SM][IM][DM_0 - DM_N] S = size of D in bytes, see enumeration as7352_item_sizes I = item id, see enumeration as7352_item_ids D = payload of the corresponding item

Note

More than one item can be string together

This function can only called if ChipLib is in state configuration.

IDs, which are readable only or not supported, will be ignored.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes).
in	*p_data*	Pointer to the data of the item.
in	*size*	Size in byte of data.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	Access to the library is blocked, call as7352_initialize at first.
ERR_DATA_TRANSFER	Communication error to sensor.
ERR_NOT_SUPPORTED	e.g. Configuration of item ITEM_ID_LED_EXT_0, but LED is not implemented.

See also

config_item_sec

4.4.2.6 as7352_get_configuration()

```
err_code_t CHIPLIB_DECLDIR as7352_get_configuration (
    const uint8_t device,
    uint8_t * p_data,
    uint32_t * p_size )
```

Request of all supported sensor and library items.

Following tasks will be done here:

- Iterate over all internal item-IDs
- Calls the internal get-function of each item-ID, if available

- Saves all data in the given data buffer

Byte sequence of data [S0][I0][D0_0 - D0_N][S1][I1][D1_0 - D1_N] ... [SM][IM][DM_0 - DM_N] S = size of D in bytes, see enumeration as7352_item_sizes I = item id, see enumeration as7352_item_ids D = payload of the corresponding item

Note

The size of the needed data buffer can be requested by calling this function with p_data = NULL and the value of p_size must be set to zero. The size will be copied to p_size.

This function can only called if ChipLib is in state configuration.

Parameters

`in`	*device*	Handle to the device (default 0, only for multi device purposes).
`out`	*p_data*	Pointer, where the item data can be saved.
`in,out`	*p_size*	Size in byte of p_data. After return of this function the used size will be saved here.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	Access to the library is blocked, call as7352_initialize at first.
ERR_DATA_TRANSFER	Communication error to sensor.
ERR_NOT_SUPPORTED	e.g. Readout of item ITEM_ID_TEMP_EXT_2, but function is not implemented.

See also

config_item_sec

4.4.2.7 as7352_start_measurement() `err_code_t CHIPLIB_DECLDIR as7352_start_measurement ( const uint8_t device )`

Starts a measurement.

This function sets only an internal event, that the measurement starts on next state machine step.

Note

This function can only called if ChipLib is in state configuration.

Parameters

`in`	*device*	Handle to the device (default 0, only for multi device purposes).

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	Access to the library is blocked, call as7352_initialize at first.
ERR_DATA_TRANSFER	Communication error to sensor.

4.4.2.8 as7352_execute_state_machine()

```
err_code_t CHIPLIB_DECLDIR as7352_execute_state_machine
(
        const uint8_t device,
        enum as7352_states * p_state )
```

Executes a measurement step.

Following tasks will be done here:

- Checks if an internal event was set.
- Checks if the internal event can be handled on actual state machine state.
- Calls the corresponding transition-function.

Note

This function can be called after finished initialization, but does only anything if the measurement was started.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes).
out	*p_state*	Pointer, where the current state can be saved, see enumeration as7352_states.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	The access to the library is blocked, call as7352_initialize at first.
ERR_DATA_TRANSFER	Communication error to sensor.
ERR_NOT_SUPPORTED	e.g. readout of item ITEM_ID_TEMP_EXT_2, but function is not implemented.
ERR_OVERFLOW	Get overflow in FIFO mode.
ERR_SENSOR_CONFIG	Break time is too high.

4.4.2.9 as7352_abort_measurement() `err_code_t CHIPLIB_DECLDIR as7352_abort_measurement ( const uint8_t device )`

Abort a measurement.

This function sets only an internal event, that the measurement stops as soon as possible.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes).

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid
ERR_PERMISSION	Access to the library is blocked, call as7352_initialize at first.

4.5 AS7352 Chip Library Functions

4.6 Error Codes

Description of given error codes.

Macros

- #define M_CHECK_ARGUMENT_LOWER_EQUAL(arg_value, max_value)
- #define M_CHECK_ARGUMENT_LOWER(arg_value, max_value)
- #define M_CHECK_ARGUMENT_MULTIPLE_OF(multiplier, value)
- #define M_CHECK_NULL_POINTER(pointer)
- #define M_CHECK_SIZE(expected, given)
- #define M_UNUSED_PARAM(x) (void)(x)

Typedefs

- typedef enum error_codes err_code_t

Enumerations

- enum error_codes {
  ERR_SUCCESS = 0,
  ERR_PERMISSION = 1,
  ERR_MESSAGE = 2,
  ERR_MESSAGE_SIZE = 3,
  ERR_POINTER = 4,
  ERR_ACCESS = 5,
  ERR_ARGUMENT = 6,
  ERR_SIZE = 7,
  ERR_NOT_SUPPORTED = 8,
  ERR_TIMEOUT = 9,
  ERR_CHECKSUM = 10,
  ERR_OVERFLOW = 11,
  ERR_EVENT = 12,
  ERR_INTERRUPT = 13,
  ERR_TIMER_ACCESS = 14,
  ERR_LED_ACCESS = 15,
  ERR_TEMP_SENSOR_ACCESS = 16,
  ERR_DATA_TRANSFER = 17,
  ERR_FIFO = 18,
  ERR_OVER_TEMP = 19,
  ERR_IDENTIFICATION = 20,
  ERR_COM_INTERFACE = 21,
  ERR_SYNCHRONISATION = 22,
  ERR_PROTOCOL = 23,
  ERR_MEMORY = 24,
  ERR_THREAD = 25,
  ERR_SPI = 26,
  ERR_DAC_ACCESS = 27,
  ERR_I2C = 28,
  ERR_NO_DATA = 29,

ERR_SYSTEM_CONFIG = 30,
ERR_USB_ACCESS = 31,
ERR_ADC_ACCESS = 32,
ERR_SENSOR_CONFIG = 33,
ERR_SATURATION = 34,
ERR_MUTEX = 35,
ERR_ACCELEROMETER = 36,
ERR_CONFIG = 37,
ERR_BLE = 38 }

4.6.1 Detailed Description

Description of given error codes.

Here you can find a generic error code table, which is used by some ams libraries.

4.6.2 Macro Definition Documentation

4.6.2.1 M_CHECK_ARGUMENT_LOWER_EQUAL

```
#define M_CHECK_ARGUMENT_LOWER_EQUAL(
        arg_value,
        max_value )
```

Value:

```
do {
        \
    if (arg_value > max_value) {
        \
        return ERR_ARGUMENT;
        \
    }
        \
} while (0)
```

Returns an error code if the arg_value is greater than the max_value.

4.6.2.2 M_CHECK_ARGUMENT_LOWER

```
#define M_CHECK_ARGUMENT_LOWER(
        arg_value,
        max_value )
```

Value:

```
do {
        \
    if (arg_value >= max_value) {
        \
        return ERR_ARGUMENT;
        \
    }
        \
} while (0)
```

Returns an error code if the first value is greater or equal than the second one.

4.6.2.3 M_CHECK_ARGUMENT_MULTIPLE_OF #define M_CHECK_ARGUMENT_MULTIPLE_OF(*multiplier,* *value*)

Value:

```
do {                                                    \
    if  ((multiplier % value) != 0) {                   \
        return ERR_ARGUMENT;                            \
    }                                                   \
} while (0)
```

Returns an error code if the value is not a multiple of multiplier.

4.6.2.4 M_CHECK_NULL_POINTER #define M_CHECK_NULL_POINTER(*pointer*)

Value:

```
do {                                                    \
    if (NULL == pointer) {                              \
        return ERR_POINTER;                             \
    }                                                   \
} while (0)
```

Returns an error code if the given address is zero.

4.6.2.5 M_CHECK_SIZE #define M_CHECK_SIZE(*expected,* *given*)

Value:

```
do {                                                    \
    if (expected != given) {                            \
        return ERR_SIZE;                                \
    }                                                   \
} while (0)
```

Returns an error code if the values are not equal.

4.6.2.6 M_UNUSED_PARAM #define M_UNUSED_PARAM(*x*) (void)(x)

Mark unused arguments to get no fails on static code analysis.

4.6.3 Typedef Documentation

4.6.3.1 err_code_t `typedef enum error_codes err_code_t`

This definition will be used for function return values.

4.6.4 Enumeration Type Documentation

4.6.4.1 error_codes `enum error_codes`

Values represent the error codes.

Enumerator

ERR_SUCCESS	Normal return code if everything was successful executed.
ERR_PERMISSION	Operation not permitted
ERR_MESSAGE	Message is invalid. For example: • Message type is not supported • incorrect crc ...
ERR_MESSAGE_SIZE	Message has the wrong size.
ERR_POINTER	Pointer is invalid. Can be a NULL Pointer or point to a wrong memory area.
ERR_ACCESS	Access denied
ERR_ARGUMENT	Invalid argument
ERR_SIZE	Argument size is too long or too short.
ERR_NOT_SUPPORTED	Function is not supported/implemented.
ERR_TIMEOUT	Got timeout while waiting for answer.
ERR_CHECKSUM	Checksum comparision failed.
ERR_OVERFLOW	Data overflow detected.
ERR_EVENT	Error to get or set an event. For example: • event queue is full or empty • receive an unexpected event ...
ERR_INTERRUPT	Error to get or set an interrupt. For example a interrupt resource is not available.
ERR_TIMER_ACCESS	Error while accessing timer periphery.
ERR_LED_ACCESS	Error while accessing LED periphery.
ERR_TEMP_SENSOR_ACCESS	Error while accessing temperature sensor.
ERR_DATA_TRANSFER	Communication error
ERR_FIFO	Faulty FIFO handling
ERR_OVER_TEMP	Overtemperature detected.
ERR_IDENTIFICATION	Sensor identification failed.

Enumerator

ERR_COM_INTERFACE	Generic communication interface error. For example: • communication interface is not available • error during open or close an communication interface ...
ERR_SYNCHRONISATION	Synchronisation error, e.g. on protocol
ERR_PROTOCOL	Generic protocol error
ERR_MEMORY	Memory allocation error
ERR_THREAD	Thread can not created.
ERR_SPI	Error while accessing SPI periphery
ERR_DAC_ACCESS	Error while accessing DAC periphery.
ERR_I2C	Error while accessing I2C periphery.
ERR_NO_DATA	No data available.
ERR_SYSTEM_CONFIG	Error during system configuration. When a system resource is not available or generates an error for example.
ERR_USB_ACCESS	USB error
ERR_ADC_ACCESS	Error while accessing ADC periphery.
ERR_SENSOR_CONFIG	Error during sensor configuration.
ERR_SATURATION	Saturation detected
ERR_MUTEX	Error while mutex handling
ERR_ACCELEROMETER	Error while reading accelerometer data
ERR_CONFIG	Software component is not fully or correctly configured
ERR_BLE	Error while executing BLE stack function

4.7 Digital I/Os

Definition for generic GPIO driver interface.

Typedefs

- typedef enum pio_modes pio_modes_t
- typedef enum pio_pulls pio_pulls_t
- typedef enum pio_states pio_states_t
- typedef void(∗ pio_callback_t) (uint32_t pio_id, pio_states_t state)

  Callback function for interrupt service routine on input pin.

Enumerations

- enum pio_modes {
  PIO_MODE_INPUT_TRIG_NONE = 0,
  PIO_MODE_INPUT_TRIG_RISING = 1,
  PIO_MODE_INPUT_TRIG_FALLING = 2,
  PIO_MODE_INPUT_TRIG_BOTH = 3,
  PIO_MODE_OUTPUT_TYPE_PP = 4,
  PIO_MODE_OUTPUT_TYPE_OD = 5,
  PIO_MODE_NUM = 6 }
- enum pio_pulls {
  PIO_PULL_NONE = 0,
  PIO_PULL_UP = 1,
  PIO_PULL_DOWN = 2,
  PIO_PULL_NUM = 3 }
- enum pio_states {
  PIO_STATE_RESET = 0,
  PIO_STATE_SET = 1,
  PIO_STATE_TOGGLE = 2,
  PIO_STATE_NUM = 3 }

Functions

- err_code_t pio_init (void ∗p_handle, pio_modes_t mode, pio_pulls_t pull, pio_callback_t p_callback)

  This function must be used to initialize the GPIO pin.
- err_code_t pio_shutdown (void ∗p_handle)

  Closes the GPIO pin.
- err_code_t pio_set (void ∗p_handle, pio_states_t state)

  Sets the state of an output pin.
- err_code_t pio_get (void ∗p_handle, pio_states_t ∗p_state)

  Gets the state of an input pin.

4.7.1 Detailed Description

Definition for generic GPIO driver interface.

This interface can be used to initialize the GPIOs for input or output purposes. Interrupt handling for inputs is implemented as well. The global device parameters and periphery connections can be commited by the handle-parameter of each function.

4.7.2 Typedef Documentation

4.7.2.1 pio_modes_t `typedef enum pio_modes pio_modes_t`

type-definition of pio_modes

4.7.2.2 pio_pulls_t `typedef enum pio_pulls pio_pulls_t`

type-definition of pio_pulls

4.7.2.3 pio_states_t `typedef enum pio_states pio_states_t`

type-definition of pio_states

4.7.2.4 pio_callback_t `typedef void(* pio_callback_t) (uint32_t pio_id, pio_states_t state)`

Callback function for interrupt service routine on input pin.

Parameters

in	*pio_id*	Identification number of the pin
in	*state*	Actual pin state, see pio_states_t

4.7.3 Enumeration Type Documentation

4.7.3.1 pio_modes `enum pio_modes`

List of supported pin configurations

Enumerator

PIO_MODE_INPUT_TRIG_NONE	normal input pin
PIO_MODE_INPUT_TRIG_RISING	input pin with interrupt on rising edge
PIO_MODE_INPUT_TRIG_FALLING	input pin with interrupt on falling edge
PIO_MODE_INPUT_TRIG_BOTH	input pin with interrupt on both edges
PIO_MODE_OUTPUT_TYPE_PP	push-pull output pin
PIO_MODE_OUTPUT_TYPE_OD	Open-Drain-Ouput
PIO_MODE_NUM	Number of supported gpio modes

4.7.3.2 pio_pulls `enum pio_pulls`

List of supported pull-up and pull-down configurations

Enumerator

PIO_PULL_NONE	No pull up/down configured.
PIO_PULL_UP	Pull-up configured.
PIO_PULL_DOWN	Pull-down configured.
PIO_PULL_NUM	Number of supported pull configurations

4.7.3.3 pio_states `enum pio_states`

List of supported pin states

Enumerator

PIO_STATE_RESET	Pin is in reset-state (0).
PIO_STATE_SET	Pin is set (1).
PIO_STATE_TOGGLE	Pin shall be toggled (1 -> 0, 0 -> 1)
PIO_STATE_NUM	Number of supported pin states

4.7.4 Function Documentation

4.7.4.1 pio_init() `err_code_t pio_init (`

```
void * p_handle,
```

```
        pio_modes_t mode,
        pio_pulls_t pull,
        pio_callback_t p_callback )
```

This function must be used to initialize the GPIO pin.

Note

This function must be called at first, otherwise all other functions return with error code ERR_PERMISSION

Parameters

in	*p_handle*	Handle to the device configuration
in	*mode*	GPIO mode see pio_modes
in	*pull*	Configuration of pull-down or pull-up, see pio_pulls
in	*p_callback*	This parameter can be used to register a callback function for interrupt handling. If not used, set to NULL.

Return values

ERR_SUCCESS	Function returns without error.
ERR_POINTER	NULL-pointer detected
ERR_ARGUMENT	An argument is invalid.

4.7.4.2 pio_shutdown() `err_code_t pio_shutdown (`

```
        void * p_handle )
```

Closes the GPIO pin.

Parameters

in	*p_handle*	Handle to the device configuration

Return values

ERR_SUCCESS	Function returns without error.
ERR_POINTER	NULL-pointer detected
ERR_ARGUMENT	An argument is invalid.

4.7.4.3 pio_set() `err_code_t pio_set (`

```
        void * p_handle,
        pio_states_t state )
```

Sets the state of an output pin.

Parameters

in	*p_handle*	Handle to the device configuration
in	*state*	New state of the pin, see pio_states

Return values

ERR_SUCCESS	Function returns without error.
ERR_POINTER	NULL-pointer detected
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	Access to the function is blocked, call pio_init first.

4.7.4.4 pio_get()

```
err_code_t pio_get (
        void * p_handle,
        pio_states_t * p_state )
```

Gets the state of an input pin.

Parameters

in	*p_handle*	Handle to the device configuration
out	*p_state*	Actual state of the pin, see pio_states

Return values

ERR_SUCCESS	Function returns without error.
ERR_POINTER	NULL-pointer detected
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	Access to the function is blocked, call pio_init first.

4.8 SPI

Definition for generic SPI master driver interface.

Typedefs

- typedef enum spi_modes spi_modes_t
- typedef enum spi_first_bit spi_first_bit_t

Enumerations

- enum spi_modes {
  SPI_MODE_PHASE_1EDGE_POL_LOW = 0,
  SPI_MODE_PHASE_1EDGE_POL_HIGH = 1,
  SPI_MODE_PHASE_2EDGE_POL_LOW = 2,
  SPI_MODE_PHASE_2EDGE_POL_HIGH = 3,
  SPI_MODE_NUM = 4 }
- enum spi_first_bit {
  SPI_FIRST_BIT_MSB = 0,
  SPI_FIRST_BIT_LSB = 1,
  SPI_FIRST_BIT_NUM = 2 }

Functions

- err_code_t spi_init (void *p_handle, uint32_t frequency, spi_modes_t mode, spi_first_bit_t first_bit)

  This function must be used to initialize the SPI periphery.
- err_code_t spi_transfer (void *p_handle, uint8_t *p_send_data, uint16_t send_len, uint8_t *p_recv_data, uint16_t recv_len)

  Transfers a SPI sequence.
- err_code_t spi_shutdown (void *p_handle)

  Closes the SPI periphery.

4.8.1 Detailed Description

Definition for generic SPI master driver interface.

This interface can be used to initialize SPI periphery and send or receive data. The global device parameters and periphery connections can be commited by the handle-parameter of each function.

4.8.2 Typedef Documentation

4.8.2.1 spi_modes_t `typedef enum spi_modes spi_modes_t`

type-definition of pio_modes

4.8.2.2 spi_first_bit_t `typedef enum spi_first_bit spi_first_bit_t`

type-definition of pio_modes

4.8.3 Enumeration Type Documentation

4.8.3.1 spi_modes `enum spi_modes`

List of supported spi modes

Enumerator

SPI_MODE_PHASE_1EDGE_POL_LOW	Phase first edge, Polarity: low
SPI_MODE_PHASE_1EDGE_POL_HIGH	Phase first edge, Polarity: high
SPI_MODE_PHASE_2EDGE_POL_LOW	Phase second edge, Polarity: low
SPI_MODE_PHASE_2EDGE_POL_HIGH	Phase second edge, Polarity: high
SPI_MODE_NUM	Number of supported spi modes

4.8.3.2 spi_first_bit `enum spi_first_bit`

List of supported first bit declarations

Enumerator

SPI_FIRST_BIT_MSB	Phase first edge, Polarity: low
SPI_FIRST_BIT_LSB	Phase first edge, Polarity: high
SPI_FIRST_BIT_NUM	Number of first bit declarations

4.8.4 Function Documentation

4.8.4.1 spi_init()

```
err_code_t spi_init (
    void * p_handle,
    uint32_t frequency,
    spi_modes_t mode,
    spi_first_bit_t first_bit )
```

This function must be used to initialize the SPI periphery.

Following tasks will be done here:

- Configuration of the corresponding pins
- Configuration of the bus frequency, SPI mode and MSB/LSB

Note

This function must be called at first, otherwise all other functions return with error code ERR_PERMISSION

Parameters

in	*p_handle*	Handle to the device configuration
in	*frequency*	Frequency of SPI-bus
in	*mode*	Configuration SPI mode, see spi_modes_t
in	*first_bit*	Configuration of the first bit, see spi_first_bit_t

Return values

ERR_SUCCESS	Function returns without error.
ERR_POINTER	NULL-pointer detected
ERR_ARGUMENT	An argument is invalid.
ERR_SPI	Error during SPI initialization

4.8.4.2 spi_transfer()

```
err_code_t spi_transfer (
    void * p_handle,
    uint8_t * p_send_data,
    uint16_t send_len,
    uint8_t * p_recv_data,
    uint16_t recv_len )
```

Transfers a SPI sequence.

Note

This function can only called if spi_init was called before

Parameters

in	*p_handle*	Handle to the device configuration
in	*p_send_data*	Pointer to send data buffer
in	*send_len*	Number of bytes to send
out	*p_recv_data*	Pointer to the memory, where received data can be saved
in	*recv_len*	Length of the requested data

Return values

ERR_SUCCESS	Function returns without error.
ERR_POINTER	NULL-pointer detected
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	Access to the function is blocked, call spi_init first.
ERR_SPI	Communication error to slave device.

4.8.4.3 spi_shutdown() `err_code_t spi_shutdown ( void * p_handle )`

Closes the SPI periphery.

Parameters

in	*p_handle*	Handle to the device configuration

Return values

ERR_SUCCESS	Function returns without error.
ERR_POINTER	NULL-pointer detected
ERR_SPI	Error while closing the SPI periphery

5 Data Structure Documentation

5.1 as7352_auto_gain Struct Reference

Data Fields

- uint8_t lower_limit
- uint8_t upper_limit

5.1.1 Detailed Description

Configuration of the auto gain algorithm

5.1.2 Field Documentation

5.1.2.1 lower_limit `uint8_t as7352_auto_gain::lower_limit`

See enumeration as7352_gain

5.1.2.2 upper_limit `uint8_t as7352_auto_gain::upper_limit`

See enumeration as7352_gain

5.2 as7352_led_config Struct Reference

Data Fields

- uint16_t enable
- uint16_t brightness

5.2.1 Detailed Description

Configuration of the LEDs with help of ITEM_ID_LED_INTERN, ITEM_ID_LED_EXT_0 ...

5.2.2 Field Documentation

5.2.2.1 enable `uint16_t as7352_led_config::enable`

Unequal zero means that LED shall be enabled, otherwise disabled.

5.2.2.2 brightness `uint16_t as7352_led_config::brightness`

Brightness value in per mile (0 - 1000)

5.3 as7352_led_pattern Struct Reference

Data Fields

- uint8_t count
- uint8_t config

5.3.1 Detailed Description

Definition for ITEM_ID_LED_PATTERN. This is used to switch the LEDs synchronized to the spectral measurement. During the measurement, the LED can only be switched on or off. The current must be configured with as7352_led_config.

5.3.2 Field Documentation

5.3.2.1 count `uint8_t as7352_led_pattern::count`

Defines how often the current LED configuration will be used.

5.3.2.2 config `uint8_t as7352_led_pattern::config`

Select the LEDs for the next measurement cycles. See as7352_led_masks.

5.4 as7352_serial Struct Reference

Data Fields

- uint32_t timestamp
- uint32_t id

5.4.1 Detailed Description

Payload definition of item ITEM_ID_SERIAL

5.4.2 Field Documentation

5.4.2.1 timestamp `uint32_t as7352_serial::timestamp`

unix timestamp of manufacturing time

5.4.2.2 id `uint32_t as7352_serial::id`

For parallel production, this ID makes the timestamp unique.

5.5 as7352_version Struct Reference

Data Fields

- uint8_t major
- uint8_t minor
- uint8_t patch
- uint8_t build

5.5.1 Detailed Description

Payload definition of item ITEM_ID_VERSION

5.5.2 Field Documentation

5.5.2.1 major `uint8_t as7352_version::major`

first position of version data: major version number

5.5.2.2 minor `uint8_t as7352_version::minor`

second position of version data: minor version number

5.5.2.3 patch `uint8_t as7352_version::patch`

third position of version data: patch version number

5.5.2.4 build `uint8_t as7352_version::build`

fourth position of version data: build version number

5.6 cmd_chiplib_sync_pwm_t Struct Reference

Data Fields

- uint8_t **enable**
- uint8_t **reserved** [3]
- uint32_t **out_0_active_time_ns**
- uint32_t **out_1_active_time_ns**
- uint32_t **sync_delay_ns**
- uint32_t **period_us**

5.6.1 Detailed Description

Payload structure for CMD_ID_CHIPLIB_SYNC_PWM

5.7 i2c_spi_config Struct Reference

Data Fields

- uint8_t enable
- uint8_t mode
- uint8_t first_bit
- uint8_t reserved
- uint32_t frequency

5.7.1 Detailed Description

Configuration structure for I2C and SPI periphery

5.7.2 Field Documentation

5.7.2.1 enable `uint8_t i2c_spi_config::enable`

Enable/Disable the periphery

5.7.2.2 mode `uint8_t i2c_spi_config::mode`

SPI only, see enumeration spi_modes

5.7.2.3 first_bit `uint8_t i2c_spi_config::first_bit`

SPI only, see enumeration spi_first_bit

5.7.2.4 reserved `uint8_t i2c_spi_config::reserved`

not used, set to zero (padding byte)

5.7.2.5 frequency `uint32_t i2c_spi_config::frequency`

Bus frequency of the periphery

5.8 i2c_xfer Struct Reference

Data Fields

- uint8_t dev_addr
- uint8_t recv_size
- uint8_t send_data [1]

5.8.1 Detailed Description

I2C transfer data

5.8.2 Field Documentation

5.8.2.1 dev_addr `uint8_t i2c_xfer::dev_addr`

I2C slave device address

5.8.2.2 recv_size `uint8_t i2c_xfer::recv_size`

Number of bytes which shall be read from the bus

5.8.2.3 send_data `uint8_t i2c_xfer::send_data[1]`

First byte of the transmit-data

5.9 i2c_xfer_16bit Struct Reference

Data Fields

- uint8_t dev_addr
- uint8_t reserved
- uint16_t recv_size
- uint8_t send_data [ ]

5.9.1 Detailed Description

I2C transfer data

5.9.2 Field Documentation

5.9.2.1 dev_addr `uint8_t i2c_xfer_16bit::dev_addr`

I2C slave device address

5.9.2.2 reserved `uint8_t i2c_xfer_16bit::reserved`

not used, set to zero (padding byte)

5.9.2.3 recv_size `uint16_t i2c_xfer_16bit::recv_size`

Number of bytes which shall be read from the bus

5.9.2.4 send_data `uint8_t i2c_xfer_16bit::send_data[]`

Begin of the transmit data

5.10 pio_config Struct Reference

Data Fields

- uint8_t enable
- uint8_t mode
- uint8_t pull

5.10.1 Detailed Description

Configuration structure for the initialization of the GPIO module

5.10.2 Field Documentation

5.10.2.1 enable `uint8_t pio_config::enable`

Enable/Disable the pin

5.10.2.2 mode `uint8_t pio_config::mode`

see enumeration pio_modes

5.10.2.3 pull `uint8_t pio_config::pull`

see enumeration pio_pulls

5.11 pwm_config Struct Reference

Data Fields

- uint8_t enable
- uint8_t reserved
- uint16_t duty_cycle
- uint32_t frequency

5.11.1 Detailed Description

Configuration structure for the initialization of the PWM pin

5.11.2 Field Documentation

5.11.2.1 enable `uint8_t pwm_config::enable`

Enable/Disable the PWM pin

5.11.2.2 reserved `uint8_t pwm_config::reserved`

not used, set to zero (padding byte)

5.11.2.3 duty_cycle `uint16_t pwm_config::duty_cycle`

duty cycle [0 - 1000]

5.11.2.4 frequency `uint32_t pwm_config::frequency`

PWM freqency in hertz

5.12 test_req Struct Reference

Data Fields

- uint16_t count
- uint16_t size
- uint32_t delay_us

5.12.1 Detailed Description

Structure for test message stream request

5.12.2 Field Documentation

5.12.2.1 count `uint16_t test_req::count`

Number of test messages to send

5.12.2.2 size `uint16_t test_req::size`

Size of test message payload

5.12.2.3 delay_us `uint32_t test_req::delay_us`

Minimum time between the transmission of two test messages in microseconds

Index