WIREPAS

Things connected – Naturally

Ruuvi sensor application for WM EVK - 1.1A

Reference Manual

Version: 1.1A

Application note

Confidential

Table of Contents

1 Ruuvi sensor application for WM EVK

1.1 Introduction

This application serves as an example on how to transport sensor data on top of Wirepas Mesh (WM) network and is designed for use with Ruuvi Tags.

This application must run on a Ruuvi tag due to the sensors it interacts with. Apart from that, the application relies on WM single MCU primitives which follow the radio and platform agnostic philosophy of WM.

1.2 Functional overview

This application queries measurement from sensors (environmental and accelerometer) on Ruuvi boards and send them at regular interval toward the WM Sink.

Data sent to the Sink is encoded as Type Length Value (TLV)) format. This application decides which sensors to read based on the configuration present in the appconfig.

1.3 Modules

The application is structured in several modules:

app.c: this is the application entry point and it manages the sensors state machine.

app_config.c: contains the configiguration structure which is initialized to default values and might change on runtime according to appconfig messages.

bme280_wrapper.c: contains the wrapper on top of BME280 Bosh driver from Github (https://github.com/BoschSensortec/BME280_driver.git).

lis2dh12_wrapper.c: contains the wrapper on top of LIS2DH12 STMicroelectronics driver from Github (https://github.com/STMicroelectronics/STMems_Standard_C_drivers.git).

format_data.c: handles packet data encoding before sending it towards the Sink.

1.4 Application datagrams

On each reading period, the application sends active sensor measurements towards the Sink. Packets are sent from endpoint 11 to endpoint 11.

The APDU is a succession of sensors measurement formated to TLV format. The sensor data is only present in the payload if it has been enabled through the configuration.

The formated data looks like this :

```
[Type - 1byte]      [Length -  1byte] [Value - N bytes little endian]
[0x01: Counter]     [0x02]            [uint16_t counter]
[0x02: Temperature] [0x04]            [int32_t  temperature]
[0x03: Humidity]    [0x04]            [uint32_t humidity]
[0x04: Pressure]    [0x04]            [uint32_t pressure]
[0x05: Accel X]     [0x04]            [int32_t  accel X]
[0x06: Accel Y]     [0x04]            [int32_t  accel Y]
[0x07: Accel Z]     [0x04]            [int32_t  accel Z]
```

The sensors range and format are the following:

```
Counter : [0-65535], incremented by 1 each period.
Temperature : [-40;+85°C], 1LSB is 0.01°C in 2's complement.
Humidity : [0;100%], 1LSB is 1/1024 % of relative humidity (1% is 1024).
Pressure : [300;1100 hPa], 1LSB is 0.01 Pascal.
Accel on X axis : [-2g;+2g], 1LSB is 1mg in 2's complement.
Accel on Y axis : [-2g;+2g], 1LSB is 1mg in 2's complement.
Accel on Z axis : [-2g;+2g], 1LSB is 1mg in 2's complement.
```

For example, if only temperature and acceleration on X axis are enabled, the payload will look like this :

```
[0x01][0x02][Counter 2Bytes][0x02][0x04][Temp 4Bytes][0x05][0x04][X accel 4Bytes]
```

1.5 Application configuration

By default the application is configured to send the measurement off all sensors every 10 seconds. This can be configured using the appconfig mechanism offered by the Wirepas stack.

The appconfig commands are formated using TLV format. Any combination of valid commands can be added to the appconfig payload.

Selecting sensor data - SENSORS_ENABLE

Each sensor can be enabled/disabled individually through the SENSORS_ENABLE command. If a sensor is disabled it does not appear in the data measurement ADPU.

The SENSORS_ENABLE command has the following format :

```
[Type - 1byte] [Length - 1byte] [Value - 6bytes]
[0x01]         [0x06]           [Temp][Humi][Press][Acc X][Acc Y][Acc Z]
```

Value 1 enables the sensor; Value 0 disables it.

For example to enable the 3 accelerometer axis and disable other sensors, the SENSORS_ENABLE command will look like this:

```
[Type - 1byte] [Length -  1byte] [Value - 6bytes]
[0x01]         [0x06]            [0x00][0x00][0x00][0x01][0x01][0x01]
```

Modifying sensor rate - SENSORS_PERIOD

The SENSORS_PERIOD command can be used to configure at what rate sensors measurement are made and data sent towards the Sink.

The SENSORS_PERIOD command has the following format :

```
[Type - 1byte] [Length - 1byte] [Value - 1-2 bytes]
[0x02]         [0x01 or 0x02]   [sensor rate (little endian)]
```

For example to set the sensors rate to 5 minutes, the SENSORS_PERIOD command will look like this:

```
[Type - 1byte] [Length -  1byte] [Value - 2 bytes]
[0x02]         [0x02]            [0x2C][0x01]
```

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Data Structure Documentation

4.1 app_config_t Struct Reference

Structure containing the application configuration.

`#include <app_config.h>`

Data Fields

- bool temperature_enable
- bool humidity_enable
- bool pressure_enable
- bool accel_x_enable
- bool accel_y_enable
- bool accel_z_enable
- uint32_t sensors_period_ms

Detailed Description

Structure containing the application configuration.

Definition at line 15 of file app_config.h.

Field Documentation

accel_x_enable

```
bool accel_x_enable
```

Enable X-axis acceleration sensor.

Definition at line 20 of file app_config.h.

accel_y_enable

```
bool accel_y_enable
```

Enable Y-axis acceleration sensor.

Definition at line 21 of file app_config.h.

accel_z_enable

```
bool accel_z_enable
```

Enable Z-axis acceleration sensor.

Definition at line 22 of file app_config.h.

humidity_enable

```
bool humidity_enable
```

Enable humidity sensor.

Definition at line 18 of file app_config.h.

pressure_enable

```
bool pressure_enable
```

Enable pressure sensor.

Definition at line 19 of file app_config.h.

sensors_period_ms

```
uint32_t sensors_period_ms
```

Period in ms at wich rate sensor data are sent to the Sink.

Definition at line 23 of file app_config.h.

temperature_enable

```
bool temperature_enable
```

Enable temperature sensor.

Definition at line 17 of file app_config.h.

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

- include/app_config.h

4.2 bme280_wrapper_measurement_t Struct Reference

Structure containing sensor measurements.

```
#include <bme280_wrapper.h>
```

Data Fields

- int32_t temperature
- uint32_t humidity
- uint32_t pressure

Detailed Description

Structure containing sensor measurements.

Definition at line 18 of file bme280_wrapper.h.

Field Documentation

humidity

```
uint32_t humidity
```

Humidity in 1/1024 % relative humidity (1% is 1024).

Definition at line 21 of file bme280_wrapper.h.

pressure

```
uint32_t pressure
```

Pressure in 0.01 Pascal.

Definition at line 23 of file bme280_wrapper.h.

temperature

```
int32_t temperature
```

Temperature in 0.01 ℃.

Definition at line 20 of file bme280_wrapper.h.

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

- include/bme280_wrapper.h

4.3 lis2dh12_wrapper_measurement_t Struct Reference

Structure containing acceleration measurements.

```
#include <lis2dh12_wrapper.h>
```

Data Fields

- int32_t accel_x
- int32_t accel_y
- int32_t accel_z

Detailed Description

Structure containing acceleration measurements.

Definition at line 17 of file lis2dh12_wrapper.h.

Field Documentation

accel_x

```
int32_t accel_x
```

Acceleration on X axis in mg [-2g / +2g].

Definition at line 19 of file lis2dh12_wrapper.h.

accel_y

```
int32_t accel_y
```

Acceleration on Y axis in mg [-2g / +2g].

Definition at line 20 of file lis2dh12_wrapper.h.

accel_z

```
int32_t accel_z
```

Acceleration on Z axis in mg [-2g / +2g].

Definition at line 21 of file lis2dh12_wrapper.h.

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

- include/lis2dh12_wrapper.h

4.4 sensor_data_t Struct Reference

This structure contains all the sensors data to be sent.

```
#include <format_data.h>
```

Data Fields

- uint16_t count
- int32_t temp
- uint32_t humi
- uint32_t press
- int32_t acc_x
- int32_t acc_y
- int32_t acc_z

Detailed Description

This structure contains all the sensors data to be sent.

Definition at line 15 of file format_data.h.

Field Documentation

acc_x

```
int32_t acc_x
```

Acceleration on X axis in mg [-2g / +2g].

Definition at line 21 of file format_data.h.

acc_y

```
int32_t acc_y
```

Acceleration on Y axis in mg [-2g / +2g].

Definition at line 22 of file format_data.h.

acc_z

```
int32_t acc_z
```

Acceleration on Z axis in mg [-2g / +2g].

Definition at line 23 of file format_data.h.

count

```
uint16_t count
```

Counter; incremented by one each period.

Definition at line 17 of file format_data.h.

humi

`uint32_t humi`

Humidity in 1/1024 % relative humidity (1% is 1024).

Definition at line 19 of file format_data.h.

press

`uint32_t press`

Pressure in 0.01 Pascal.

Definition at line 20 of file format_data.h.

temp

`int32_t temp`

Temperature in 0.01 ℃.

Definition at line 18 of file format_data.h.

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

- include/format_data.h

5 File Documentation

5.1 app.c File Reference

Reference application of the Ruuvi evaluation kit.

```
#include <stdio.h>
#include <string.h>
#include "api.h"
#include "node_configuration.h"
#include "tlv.h"
#include "app_scheduler.h"
#include "board.h"
#include "spi.h"
#include "bme280_wrapper.h"
#include "lis2dh12_wrapper.h"
#include "app_config.h"
#include "format_data.h"
```

Macros

- #define SENSOR_DATA_DST_ENDPOINT 11
- #define SENSOR_DATA_SRC_ENDPOINT 11

Enumerations

- enum sensor_task_state_e { SENSOR_TASK_STATE_START_MEAS, SENSOR_TASK_STA↩ TE_SEND_DATA }

Functions

- static bool ruuvi_spi_init (void)

  initialize SPI driver and sensors chip select pins.
- static void send_data (sensor_data_t ∗data)

  Sends the sensors data.
- static uint32_t sensor_task ()

  Sensor task. Manage the sensors and sends the measurement to the Sink.
- static void on_config_update (void)

  Function called when the application configuration as changed.
- void App_init (const app_global_functions_t ∗functions)

  Initialization callback for application.

Variables

- sensor_task_state_e m_task_state
- sensor_data_t m_sensor_data

Detailed Description

Reference application of the Ruuvi evaluation kit.

Copyright

Wirepas Oy 2019

Macro Definition Documentation

SENSOR_DATA_DST_ENDPOINT

```
#define SENSOR_DATA_DST_ENDPOINT 11
```

Definition at line 23 of file app.c.

SENSOR_DATA_SRC_ENDPOINT

```
#define SENSOR_DATA_SRC_ENDPOINT 11
```

Definition at line 24 of file app.c.

Enumeration Type Documentation

sensor_task_state_e

```
enum sensor_task_state_e
```

Enumerator

SENSOR_TASK_STATE_START_MEAS	Start the sensor measurements.
SENSOR_TASK_STATE_SEND_DATA	Read the measurement and send them.

Definition at line 27 of file app.c.

```
28 {
29     SENSOR_TASK_STATE_START_MEAS,
30     SENSOR_TASK_STATE_SEND_DATA
31 } sensor_task_state_e;
```

Function Documentation

App_init()

```
void App_init (
               const app_global_functions_t * functions )
```

Initialization callback for application.

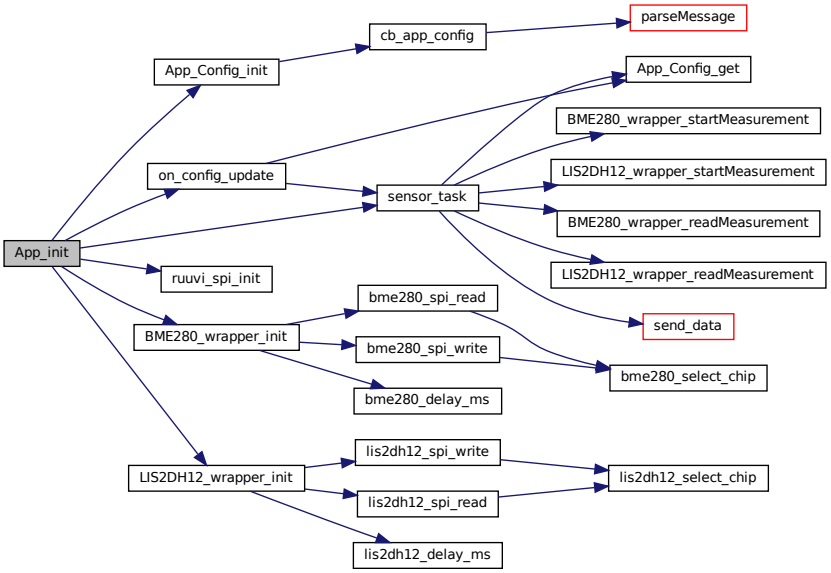
This function is called after hardware has been initialized but the stack is not yet running.

Definition at line 211 of file app.c.

```
212 {
213     /* Open public API. */
214     API_Open(functions);
215
216     /* Basic configuration of the node with a unique node address. */
217     if (configureNode(getUniqueAddress(),
218                       NETWORK_ADDRESS,
219                       NETWORK_CHANNEL) != APP_RES_OK)
220     {
221         /*
222          * Could not configure the node.
223          * It should not happen except if one of the config value is invalid.
224          */
225         return;
```

```
226     }
227
228     m_sensor_data.count = 0;
229     m_task_state = SENSOR_TASK_STATE_START_MEAS;
230
231     /* Initialize all the modules. */
232     App_Config_init(on_config_update);
233     App_Scheduler_init();
234     ruuvi_spi_init();
235     BME280_wrapper_init();
236     LIS2DH12_wrapper_init();
237
238     /* Launch the sensor task. */
239     App_Scheduler_addTask(sensor_task, APP_SCHEDULER_SCHEDULE_ASAP);
240
241     /*
242      * Start the stack.
243      * This is really important step, otherwise the stack will stay stopped and
244      * will not be part of any network. So the device will not be reachable
245      * without reflashing it
246      */
247     lib_state->startStack();
248 }
```

Here is the call graph for this function:



on_config_update()

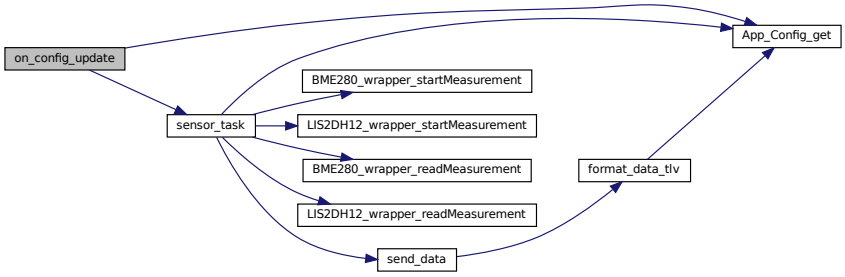
```
static void on_config_update (
              void  )  [static]
```

Function called when the application configuration as changed.

Definition at line 189 of file app.c.

```
190 {
191     const app_config_t * cfg = App_Config_get();
192
193     if(cfg->sensors_period_ms == 0)
194     {
195         /* Cancel sensor task. Don't send sensor data anymore. */
196         App_Scheduler_cancelTask(sensor_task);
197     }
198     else
199     {
200         /* Config changed, schedule Sensor task ASAP. */
201         App_Scheduler_addTask(sensor_task, APP_SCHEDULER_SCHEDULE_ASAP);
202     }
203 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

ruuvi_spi_init()

```
static bool ruuvi_spi_init (
               void  )  [static]
```

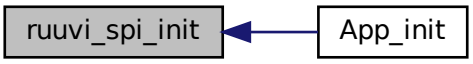
initialize SPI driver and sensors chip select pins.

Definition at line 41 of file app.c.

```
42 {
43     spi_res_e res;
44     spi_conf_t conf;
45
46     /* Initialize LIS2DH12 Chip select pin. */
47     nrf_gpio_pin_dir_set(BOARD_SPI_LIS2DH12_CS_PIN, NRF_GPIO_PIN_DIR_OUTPUT);
48     nrf_gpio_cfg_output(BOARD_SPI_LIS2DH12_CS_PIN);
```

```
49      nrf_gpio_pin_set(BOARD_SPI_LIS2DH12_CS_PIN);
50
51      /* Initialize BME280 Chip select pin. */
52      nrf_gpio_pin_dir_set(BOARD_SPI_BME280_CS_PIN, NRF_GPIO_PIN_DIR_OUTPUT);
53      nrf_gpio_cfg_output(BOARD_SPI_BME280_CS_PIN);
54      nrf_gpio_pin_set(BOARD_SPI_BME280_CS_PIN);
55
56      /* Initialize SPI driver. */
57      conf.bit_order = SPI_ORDER_MSB;
58      conf.clock = 4000000;
59      conf.mode = SPI_MODE_HIGH_FIRST;
60      res = SPI_init(&conf);
61      if ((res != SPI_RES_OK) && (res != SPI_RES_ALREADY_INITIALIZED))
62      {
63          return false;
64      }
65
66      return true;
67  }
```

Here is the caller graph for this function:



send_data()

```
static void send_data (
               sensor_data_t * data )  [static]
```

Sends the sensors data.

Parameters

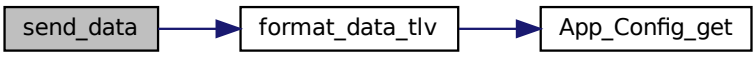
in	*data*	Pointer to the structure containing sensor data to send.

Definition at line 74 of file app.c.

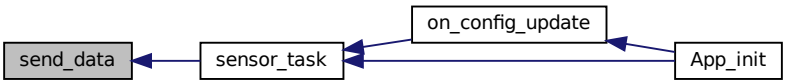
```
75  {
76      size_t count = 0;
77      uint8_t buff[102];
78      int len = 102;
79
80      /* For now the only supported format is TLV */
81      len = format_data_tlv(buff, &m_sensor_data, len);
82
83      /* Only send data if there is a route to the Sink. */
84      app_res_e res = lib_state->getRouteCount(&count);
85      if (res == APP_RES_OK && count > 0 && len != -1)
86      {
87          app_lib_data_to_send_t data_to_send;
88          data_to_send.bytes = (const uint8_t *) buff;
89          data_to_send.num_bytes = len;
90          data_to_send.dest_address = APP_ADDR_ANYSINK;
91          data_to_send.src_endpoint = SENSOR_DATA_SRC_ENDPOINT;
92          data_to_send.dest_endpoint = SENSOR_DATA_DST_ENDPOINT;
```

```
93          data_to_send.qos = APP_LIB_DATA_QOS_HIGH;
94          data_to_send.delay = 0;
95          data_to_send.flags = APP_LIB_DATA_SEND_FLAG_NONE;
96          data_to_send.tracking_id = APP_LIB_DATA_NO_TRACKING_ID;
97
98          /* Send the data packet. */
99          lib_data->sendData(&data_to_send);
100     }
101 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



sensor_task()

```
static uint32_t sensor_task ( )  [static]
```

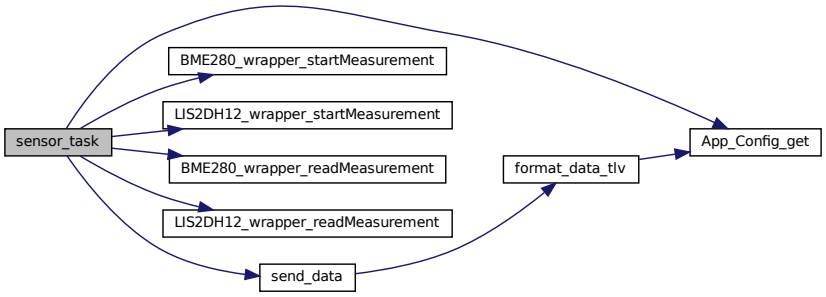
Sensor task. Manage the sensors and sends the measurement to the Sink.

Definition at line 107 of file app.c.

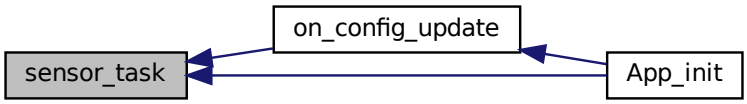
```
108 {
109     const app_config_t * cfg = App_Config_get();
110     uint32_t time_to_run;
111
112     switch (m_task_state)
113     {
114         case SENSOR_TASK_STATE_START_MEAS:
115         {
116             time_to_run = APP_SCHEDULER_SCHEDULE_ASAP;
117             m_task_state = SENSOR_TASK_STATE_SEND_DATA;
118
119             if(cfg->temperature_enable ||
120                cfg->humidity_enable ||
121                cfg->pressure_enable )
122             {
123                 time_to_run = BME280_wrapper_startMeasurement();
124             }
125
126             if(cfg->accel_x_enable ||
127                cfg->accel_y_enable ||
128                cfg->accel_z_enable )
129             {
```

```
130                 uint32_t lis2dh12_time_to_run;
131                 lis2dh12_time_to_run = LIS2DH12_wrapper_startMeasurement()
    ;
132
133                 if(lis2dh12_time_to_run > time_to_run)
134                 {
135                     time_to_run = lis2dh12_time_to_run;
136                 }
137             }
138             break;
139         }
140         case SENSOR_TASK_STATE_SEND_DATA:
141         {
142             /* Counter is always enabled. Increment it each period. */
143             m_sensor_data.count++;
144
145             if (cfg->temperature_enable ||
146                 cfg->humidity_enable ||
147                 cfg->pressure_enable )
148             {
149                 bme280_wrapper_measurement_t measurement;
150                 BME280_wrapper_readMeasurement(&measurement);
151
152                 m_sensor_data.temp = measurement.temperature;
153                 m_sensor_data.press = measurement.pressure;
154                 m_sensor_data.humi = measurement.humidity;
155             }
156
157             if(cfg->accel_x_enable ||
158                cfg->accel_y_enable ||
159                cfg->accel_z_enable )
160             {
161                 lis2dh12_wrapper_measurement_t measurement;
162                 LIS2DH12_wrapper_readMeasurement(&measurement);
163
164                 m_sensor_data.acc_x = measurement.accel_x;
165                 m_sensor_data.acc_y = measurement.accel_y;
166                 m_sensor_data.acc_z = measurement.accel_z;
167             }
168
169             send_data(&m_sensor_data);
170
171             m_task_state = SENSOR_TASK_STATE_START_MEAS;
172             time_to_run = cfg->sensors_period_ms;
173             break;
174         }
175         default:
176         {
177             m_task_state = SENSOR_TASK_STATE_START_MEAS;
178             time_to_run = cfg->sensors_period_ms;
179             break;
180         }
181     }
182
183     return time_to_run;
184 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



Variable Documentation

m_sensor_data

```
sensor_data_t m_sensor_data
```

Definition at line 36 of file app.c.

m_task_state

```
sensor_task_state_e m_task_state
```

Definition at line 34 of file app.c.

5.2 app_config.c File Reference

```
#include <string.h>
#include "api.h"
#include "app_config.h"
#include "tlv.h"
```

Macros

- #define DEFAULT_TEMPERATURE_EN true
- #define DEFAULT_HUMIDITY_EN true
- #define DEFAULT_PRESSURE_EN true
- #define DEFAULT_ACCEL_X_EN true
- #define DEFAULT_ACCEL_Y_EN true
- #define DEFAULT_ACCEL_Z_EN true
- #define DEFAULT_SENSORS_PERIOD_MS (10∗1000)
- #define SENSOR_PERIOD_MIN_S 1
- #define SENSOR_PERIOD_MAX_S 3600
- #define SENSOR_ENABLE_VALUE 1
- #define SENSOR_EN_TEMPERATURE_BYTE 0
- #define SENSOR_EN_HUMIDITY_BYTE 1
- #define SENSOR_EN_PRESSURE_BYTE 2
- #define SENSOR_EN_ACCEL_X_BYTE 3
- #define SENSOR_EN_ACCEL_Y_BYTE 4
- #define SENSOR_EN_ACCEL_Z_BYTE 5

Enumerations

- enum app_config_cmd_e { CMDID_SENSORS_ENABLE = 1, CMDID_SENSORS_PERIOD = 2 }

Functions

- static bool handleCommandSensorsEnable (tlv_item *item)
  Configure which sensor is enable.
- static bool handleCommandSensorsPeriod (tlv_item *item)
  Check that Sensor period is within boundaries before it will take it into use.
- static bool parseMessage (const uint8_t *msg, uint8_t length)
  Parse the received configuration.
- static void cb_app_config (const uint8_t *bytes, uint8_t seq, uint16_t interval)
  Wrapper for the reception of an appconfig.
- void App_Config_init (on_config_change_cb_f cb)
  Initialize the App Config module.
- const app_config_t * App_Config_get (void)
  Returns a pointer to the configuration structure.

Variables

- app_config_t m_config
- on_config_change_cb_f m_callback

Macro Definition Documentation

DEFAULT_ACCEL_X_EN

```
#define DEFAULT_ACCEL_X_EN true
```

Definition at line 12 of file app_config.c.

DEFAULT_ACCEL_Y_EN

```
#define DEFAULT_ACCEL_Y_EN true
```

Definition at line 13 of file app_config.c.

DEFAULT_ACCEL_Z_EN

```
#define DEFAULT_ACCEL_Z_EN true
```

Definition at line 14 of file app_config.c.

DEFAULT_HUMIDITY_EN

```
#define DEFAULT_HUMIDITY_EN true
```

Definition at line 10 of file app_config.c.

DEFAULT_PRESSURE_EN

```
#define DEFAULT_PRESSURE_EN true
```

Definition at line 11 of file app_config.c.

DEFAULT_SENSORS_PERIOD_MS

```
#define DEFAULT_SENSORS_PERIOD_MS (10*1000)
```

Definition at line 15 of file app_config.c.

DEFAULT_TEMPERATURE_EN

```
#define DEFAULT_TEMPERATURE_EN true
```

Definition at line 9 of file app_config.c.

SENSOR_EN_ACCEL_X_BYTE

```
#define SENSOR_EN_ACCEL_X_BYTE 3
```

Definition at line 28 of file app_config.c.

SENSOR_EN_ACCEL_Y_BYTE

```
#define SENSOR_EN_ACCEL_Y_BYTE 4
```

Definition at line 29 of file app_config.c.

SENSOR_EN_ACCEL_Z_BYTE

```
#define SENSOR_EN_ACCEL_Z_BYTE 5
```

Definition at line 30 of file app_config.c.

SENSOR_EN_HUMIDITY_BYTE

```
#define SENSOR_EN_HUMIDITY_BYTE 1
```

Definition at line 26 of file app_config.c.

SENSOR_EN_PRESSURE_BYTE

```
#define SENSOR_EN_PRESSURE_BYTE 2
```

Definition at line 27 of file app_config.c.

SENSOR_EN_TEMPERATURE_BYTE

```
#define SENSOR_EN_TEMPERATURE_BYTE 0
```

Definition at line 25 of file app_config.c.

SENSOR_ENABLE_VALUE

```
#define SENSOR_ENABLE_VALUE 1
```

Definition at line 22 of file app_config.c.

SENSOR_PERIOD_MAX_S

```
#define SENSOR_PERIOD_MAX_S 3600
```

Definition at line 20 of file app_config.c.

SENSOR_PERIOD_MIN_S

```
#define SENSOR_PERIOD_MIN_S 1
```

Definition at line 18 of file app_config.c.

Enumeration Type Documentation

app_config_cmd_e

```
enum app_config_cmd_e
```

Enumerator

CMDID_SENSORS_ENABLE	Sensor enable command.
CMDID_SENSORS_PERIOD	Set sensor period command.

Definition at line 33 of file app_config.c.

```
34 {
35     CMDID_SENSORS_ENABLE = 1,
36     CMDID_SENSORS_PERIOD = 2,
37 } app_config_cmd_e;
```

Function Documentation

App_Config_get()

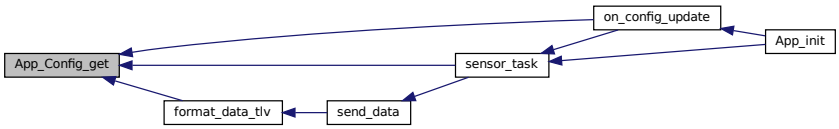
```
const app_config_t* App_Config_get (
                void  )
```

Returns a pointer to the configuration structure.

Definition at line 182 of file app_config.c.

```
183 {
184     return &m_config;
185 }
```

Here is the caller graph for this function:



App_Config_init()

```
void App_Config_init (
                on_config_change_cb_f cb )
```

Initialize the App Config module.

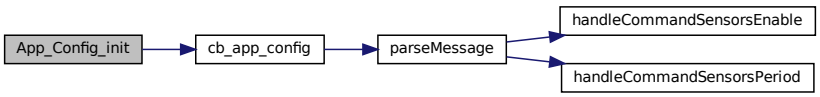
Parameters

in	*cb*	Pointer to a callback function. NULL if not used.

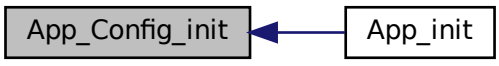
Definition at line 167 of file app_config.c.

```
168 {
169     m_config.temperature_enable =
      DEFAULT_TEMPERATURE_EN;
170     m_config.humidity_enable = DEFAULT_HUMIDITY_EN;
171     m_config.pressure_enable = DEFAULT_PRESSURE_EN;
172     m_config.accel_x_enable = DEFAULT_ACCEL_X_EN;
173     m_config.accel_y_enable = DEFAULT_ACCEL_Y_EN;
174     m_config.accel_z_enable = DEFAULT_ACCEL_Z_EN;
175     m_config.sensors_period_ms =
      DEFAULT_SENSORS_PERIOD_MS;
176
177     m_callback = cb;
178
179     lib_data->setNewAppConfigCb(cb_app_config);
180 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



cb_app_config()

```
static void cb_app_config (
               const uint8_t * bytes,
               uint8_t seq,
               uint16_t interval )  [static]
```

Wrapper for the reception of an appconfig.

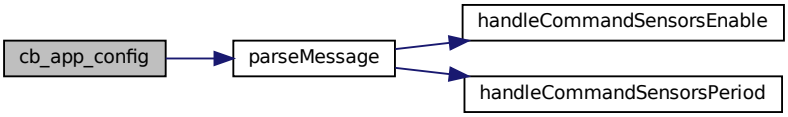
Parameters

in	*bytes*	The appconfig bytes
in	*seq*	The appconfig sequence number
in	*interval*	The appconfig interval rate (in seconds)

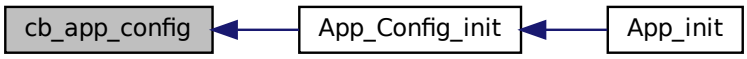
Definition at line 153 of file app_config.c.

```
154 {
155     /* Unused parameters. */
156     (void) seq;
157     (void) interval;
158 
159     parseMessage(bytes, lib_data->getAppConfigNumBytes());
160 
161     if (m_callback != NULL)
162     {
163         m_callback();
164     }
165 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



handleCommandSensorsEnable()

```
static bool handleCommandSensorsEnable (
                tlv_item * item )  [static]
```

Configure which sensor is enable.

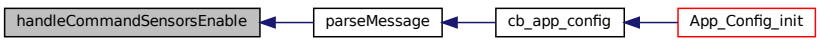
Returns

Response to the request ok or not ok.

Definition at line 48 of file app_config.c.

```
49 {
50     bool ret = false;
51
52     if (item->length == 6)
53     {
54         m_config.temperature_enable =
55             (item->value[SENSOR_EN_TEMPERATURE_BYTE] ==
      SENSOR_ENABLE_VALUE);
56         m_config.humidity_enable =
57             (item->value[SENSOR_EN_HUMIDITY_BYTE] ==
      SENSOR_ENABLE_VALUE);
58         m_config.pressure_enable =
59             (item->value[SENSOR_EN_PRESSURE_BYTE] ==
      SENSOR_ENABLE_VALUE);
60         m_config.accel_x_enable =
61             (item->value[SENSOR_EN_ACCEL_X_BYTE] ==
      SENSOR_ENABLE_VALUE);
62         m_config.accel_y_enable =
63             (item->value[SENSOR_EN_ACCEL_Y_BYTE] ==
      SENSOR_ENABLE_VALUE);
64         m_config.accel_z_enable =
65             (item->value[SENSOR_EN_ACCEL_Z_BYTE] ==
      SENSOR_ENABLE_VALUE);
66         ret = true;
67     }
68
69     return ret;
70 }
```

Here is the caller graph for this function:



handleCommandSensorsPeriod()

```
static bool handleCommandSensorsPeriod (
              tlv_item * item )  [static]
```

Check that Sensor period is within boundaries before it will take it into use.

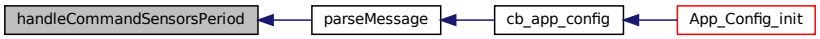
Returns

Response to the request ok or not ok.

Definition at line 77 of file app_config.c.

```
78 {
79     bool ret = false;
80 
81     if ((item->length == 1) || (item->length == 2))
82     {
83         uint16_t val = 0;
84         memcpy(&val, item->value, item->length);
85 
86         if (val >= SENSOR_PERIOD_MIN_S && val <=
      SENSOR_PERIOD_MAX_S)
87         {
88             m_config.sensors_period_ms = val*1000;
89             ret = true;
90         }
91         else if (val == 0)
92         {
93             m_config.sensors_period_ms = 0;
94             ret = true;
95         }
96     }
97 
98     return ret;
99 }
```

Here is the caller graph for this function:



parseMessage()

```
static bool parseMessage (
               const uint8_t * msg,
               uint8_t length )  [static]
```

Parse the received configuration.

Parameters

in	*msg*	Pointer to the received configuration buffer.
in	*length*	Length of the configuration buffer.

Returns

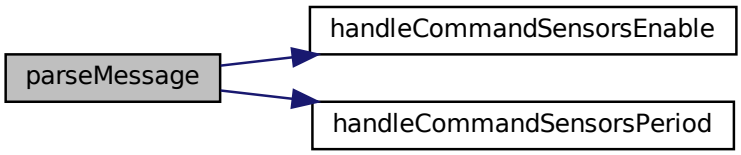
Response to the request ok or not ok.

Definition at line 107 of file app_config.c.

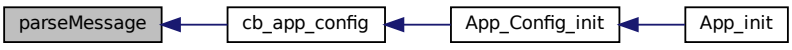
```
108 {
109     bool rval = true;
110     tlv_record record;
111 
112     Tlv_init(&record, (uint8_t *)msg, length);
113 
114     while (rval == true)
115     {
116         tlv_item * item;
```

```
117         tlv_res_e tlv_res;
118
119         tlv_res = Tlv_Decode_getNextItem(&record,&item);
120
121         if (tlv_res == TLV_RES_ERROR)
122         {
123             rval = false;
124             break;
125         }
126         else if (tlv_res == TLV_RES_END)
127         {
128             break;
129         }
130
131         switch (item->type)
132         {
133             case CMDID_SENSORS_ENABLE:
134                 rval = handleCommandSensorsEnable(item);
135                 break;
136             case CMDID_SENSORS_PERIOD:
137                 rval = handleCommandSensorsPeriod(item);
138                 break;
139             default:
140                 /* Unknown command but tlv record is valid. */
141                 break;
142         }
143     }
144     return rval;
145 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



Variable Documentation

m_callback

`on_config_change_cb_f m_callback`

Definition at line 42 of file app_config.c.

m_config

```
app_config_t m_config
```

Definition at line 40 of file app_config.c.

5.3 bme280_wrapper.c File Reference

Wrapper on top of BME280 driver from Bosch GitHub.

```
#include "bme280.h"
#include "board.h"
#include "hal_api.h"
#include "spi.h"
#include "api.h"
#include <string.h>
#include "bme280_wrapper.h"
```

Macros

- #define BME280_MEASUREMENT_TIME_MS (9.3∗1.5)
- #define MAX_WRITE_SIZE 32
- #define MAX_READ_SIZE 32

Functions

- static void bme280_delay_ms (uint32_t period)
  Blocking wait for a given amount of time.
- void bme280_select_chip (bool select)
  Select or unselect BME280 with its chip select signal.
- static int8_t bme280_spi_read (uint8_t dev_id, uint8_t reg_addr, uint8_t ∗reg_data, uint16_t len)
  Read from spi (function required by Bosh Lib).
- static int8_t bme280_spi_write (uint8_t dev_id, uint8_t reg_addr, uint8_t ∗reg_data, uint16_t len)
  Write with spi (function required by Bosh Lib).
- bool BME280_wrapper_init (void)
  Initialize the BME280 wrapper library.
- uint32_t BME280_wrapper_startMeasurement (void)
  Start a measurement.
- bool BME280_wrapper_readMeasurement (bme280_wrapper_measurement_t ∗measurement)
  Get a measurement previously asked by a start measurement.

Variables

- static struct bme280_dev m_bme280_dev

Detailed Description

Wrapper on top of BME280 driver from Bosch GitHub.

Copyright

Wirepas Oy 2019

Macro Definition Documentation

BME280_MEASUREMENT_TIME_MS

```
#define BME280_MEASUREMENT_TIME_MS (9.3*1.5)
```

Definition at line 15 of file bme280_wrapper.c.

MAX_READ_SIZE

```
#define MAX_READ_SIZE 32
```

Definition at line 21 of file bme280_wrapper.c.

MAX_WRITE_SIZE

```
#define MAX_WRITE_SIZE 32
```

Definition at line 18 of file bme280_wrapper.c.

Function Documentation

bme280_delay_ms()

```
static void bme280_delay_ms (
              uint32_t period )  [static]
```

Blocking wait for a given amount of time.

Parameters

in	*period*	Time to wait in ms.

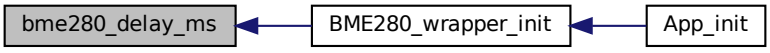
Note

Lib doesn't use it a lot and for only short period mainly at init stage executed from app_init.

Definition at line 32 of file bme280_wrapper.c.

```
33 {
34     app_lib_time_timestamp_hp_t end;
35     end = lib_time->addUsToHpTimestamp(lib_time->getTimestampHp(),
36                                        period * 1000);
37
38     /* Active wait until period is elapsed */
39     while (lib_time->isHpTimestampBefore(lib_time->getTimestampHp(),
40                                          end));
41 }
```

Here is the caller graph for this function:



bme280_select_chip()

```
void bme280_select_chip (
                bool select )
```

Select or unselect BME280 with its chip select signal.

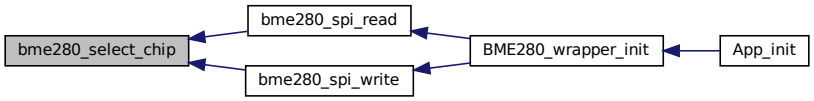
Parameters

in	*select*	True to select and false to unselect.

Definition at line 47 of file bme280_wrapper.c.

```
48 {
49     if (select)
50     {
51         nrf_gpio_pin_clear(BOARD_SPI_BME280_CS_PIN);
52     }
53     else
54     {
55         nrf_gpio_pin_set(BOARD_SPI_BME280_CS_PIN);
56     }
57 }
```

Here is the caller graph for this function:



bme280_spi_read()

```
static int8_t bme280_spi_read (
               uint8_t dev_id,
               uint8_t reg_addr,
               uint8_t * reg_data,
               uint16_t len )  [static]
```

Read from spi (function required by Bosh Lib).

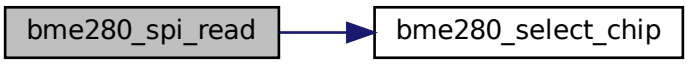
Parameters

in	*dev_id*	Device id (unused here).
in	*reg_addr*	First register to read.
in	*reg_data*	Pointer to store read registers.
in	*len*	Number of registers (of 1 byte) to read.

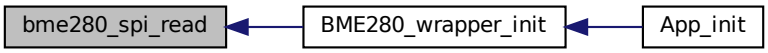
Definition at line 66 of file bme280_wrapper.c.

```
70 {
71     spi_res_e res;
72     uint8_t tx[1];
73     uint8_t rx[MAX_READ_SIZE];
74 
75     if (len > MAX_READ_SIZE)
76     {
77         return -1;
78     }
79 
80     tx[0] = reg_addr;
81 
82     spi_xfer_t transfer;
83     transfer.write_ptr = tx;
84     transfer.write_size = 1;
85     transfer.read_ptr = rx;
86     transfer.read_size = len + 1;
87 
88     bme280_select_chip(true);
89     /* Blocking read. */
90     res = SPI_transfer(&transfer, NULL);
91     bme280_select_chip(false);
92 
93     if (res != SPI_RES_OK)
94     {
95         return -1;
96     }
97 
98     memcpy(reg_data, &rx[1], len);
99     return 0;
100 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



bme280_spi_write()

```
static int8_t bme280_spi_write (
                uint8_t dev_id,
                uint8_t reg_addr,
                uint8_t * reg_data,
                uint16_t len )  [static]
```

Write with spi (function required by Bosh Lib).

Parameters

in	*dev_id*	Device id (unused here).
in	*reg_addr*	First register to write.
in	*reg_data*	Pointer to value to write.
in	*len*	Number of registers (of 1 byte) to write.

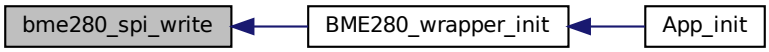
Definition at line 109 of file bme280_wrapper.c.

```
113 {
114     spi_res_e res;
115     uint8_t tx[MAX_WRITE_SIZE + 1];
116 
117     if (len > MAX_WRITE_SIZE)
118     {
119         return -1;
120     }
121 
122     tx[0] = reg_addr;
123     memcpy(&tx[1], reg_data, len);
124 
```

```
125     spi_xfer_t transfer;
126     transfer.write_ptr = tx;
127     transfer.write_size = len + 1;
128     transfer.read_ptr = NULL;
129     transfer.read_size = 0;
130
131     bme280_select_chip(true);
132     /* Blocking write. */
133     res = SPI_transfer(&transfer, NULL);
134     bme280_select_chip(false);
135
136     if (res != SPI_RES_OK)
137     {
138         return -1;
139     }
140
141     return 0;
142 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



BME280_wrapper_init()

```
bool BME280_wrapper_init (
                void  )
```

Initialize the BME280 wrapper library.

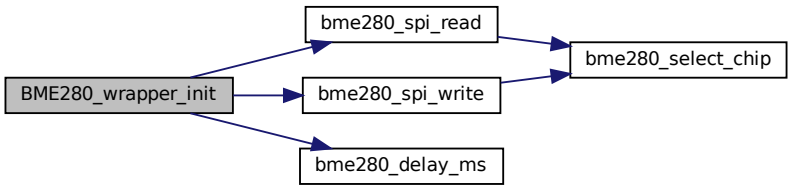
Returns

True if successfully initialized, false otherwise.

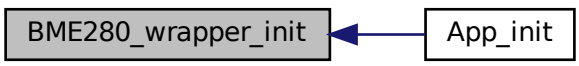
Definition at line 144 of file bme280_wrapper.c.

```
145 {
146     uint8_t settings_sel;
147
148     /* Create device for BME280 driver with the adapted read/write functions. */
149     m_bme280_dev.dev_id = 0;
150     m_bme280_dev.intf = BME280_SPI_INTF;
151     m_bme280_dev.read = bme280_spi_read;
152     m_bme280_dev.write = bme280_spi_write;
153     m_bme280_dev.delay_ms = bme280_delay_ms;
154
155     if (bme280_init(&m_bme280_dev) != BME280_OK)
156     {
157         return false;
158     }
159
160     /*
161      * Configure the mode of operation. It has a "Weather measurement" config
162      * as described in RM.
163      */
164     m_bme280_dev.settings.osr_h = BME280_OVERSAMPLING_1X;
165     m_bme280_dev.settings.osr_p = BME280_OVERSAMPLING_1X;
166     m_bme280_dev.settings.osr_t = BME280_OVERSAMPLING_1X;
167     m_bme280_dev.settings.filter = BME280_FILTER_COEFF_OFF;
168     settings_sel = BME280_OSR_PRESS_SEL |
169                    BME280_OSR_TEMP_SEL |
170                    BME280_OSR_HUM_SEL |
171                    BME280_FILTER_SEL;
172
173     if (bme280_set_sensor_settings(settings_sel, &m_bme280_dev) != BME280_OK)
174     {
175         return false;
176     }
177
178     return true;
179 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



BME280_wrapper_readMeasurement()

```
bool BME280_wrapper_readMeasurement (
              bme280_wrapper_measurement_t * measurement )
```

Get a measurement previously asked by a start measurement.

Parameters

in	*measurement*	The measurement read out from the sensor.

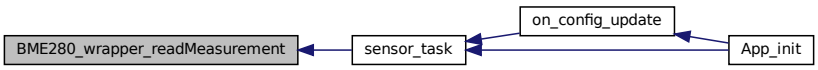
Returns

false if a problem occured, true otherwise.

Definition at line 196 of file bme280_wrapper.c.

```
197 {
198     struct bme280_data comp_data;
199     int8_t rslt;
200
201     rslt = bme280_get_sensor_data(BME280_ALL, &comp_data, &m_bme280_dev);
202     if (rslt != BME280_OK)
203     {
204         return false;
205     }
206
207     measurement->humidity = comp_data.humidity;
208     measurement->pressure = comp_data.pressure;
209     measurement->temperature = comp_data.temperature;
210
211     return true;
212 }
```

Here is the caller graph for this function:



BME280_wrapper_startMeasurement()

```
uint32_t BME280_wrapper_startMeasurement (
               void  )
```

Start a measurement.

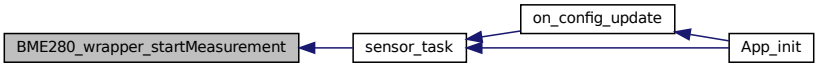
Returns

The time in ms to wait before the measurement is ready.

Definition at line 181 of file bme280_wrapper.c.

```
182 {
183     int8_t rslt;
184
185     /* Start a measurement. */
186     rslt = bme280_set_sensor_mode(BME280_FORCED_MODE, &m_bme280_dev);
187     (void) rslt;
188
189     /*
190      * According to our config, measurement max time is 9.3ms. Time can be
191      * computed according to chapter 9.3 of BME280 RM.
192      */
193     return BME280_MEASUREMENT_TIME_MS;
194 }
```

Here is the caller graph for this function:



Variable Documentation

m_bme280_dev

```
struct bme280_dev m_bme280_dev  [static]
```

Definition at line 24 of file bme280_wrapper.c.

5.4 format_data.c File Reference

This module contains functions to format data to different format before sending them.

```
#include "format_data.h"
#include "app_config.h"
#include "tlv.h"
```

Enumerations

- enum sensor_tlv_type_e {
  TLV_TYPE_COUNTER = 0x01, TLV_TYPE_TEMPERATURE = 0x02, TLV_TYPE_HUMIDITY = 0x03, TLV_TYPE_PRESSURE = 0x04,
  TLV_TYPE_ACCEL_X = 0x05, TLV_TYPE_ACCEL_Y = 0x06, TLV_TYPE_ACCEL_Z = 0x07 }
  List of sensor types for TLV packet format.

Functions

- int format_data_tlv (uint8_t *buffer, sensor_data_t *data, int length)
  Format sensor data to TLV format.

Detailed Description

This module contains functions to format data to different format before sending them.

Copyright

Wirepas Oy 2019

Enumeration Type Documentation

sensor_tlv_type_e

```
enum sensor_tlv_type_e
```

List of sensor types for TLV packet format.

Enumerator

TLV_TYPE_COUNTER	
TLV_TYPE_TEMPERATURE	
TLV_TYPE_HUMIDITY	
TLV_TYPE_PRESSURE	
TLV_TYPE_ACCEL_X	
TLV_TYPE_ACCEL_Y	
TLV_TYPE_ACCEL_Z	

Definition at line 14 of file format_data.c.

```
15 {
16     TLV_TYPE_COUNTER     = 0x01,
17     TLV_TYPE_TEMPERATURE = 0x02,
18     TLV_TYPE_HUMIDITY    = 0x03,
19     TLV_TYPE_PRESSURE    = 0x04,
20     TLV_TYPE_ACCEL_X     = 0x05,
21     TLV_TYPE_ACCEL_Y     = 0x06,
22     TLV_TYPE_ACCEL_Z     = 0x07,
23 } sensor_tlv_type_e;
```

Function Documentation

format_data_tlv()

```
int format_data_tlv (
                uint8_t * buffer,
                sensor_data_t * data,
                int len )
```

Format sensor data to TLV format.

Formatted buffer looks like this: [Type] [Length] [Value] [0x01: Counter] [0x02] [uint16_t counter] [0x02: Temperature] [0x04] [int32_t temperature] [0x03: Humidity] [0x04] [uint32_t humidity] [0x04: Pressure] [0x04] [uint32_t pressure] [0x05: Accel X] [0x04] [int32_t accel X] [0x06: Accel Y] [0x04] [int32_t accel Y] [0x07: Accel Z] [0x04] [int32_t accel Z] Sensor data is only present in the formatted packet if it as been enabled in the configuration. Counter is always present.

Parameters

out	*buffer*	Buffer to store formatted data.
in	*data*	Pointer to the structure holding sensors data.
in	*len*	Length of the buffer

Returns

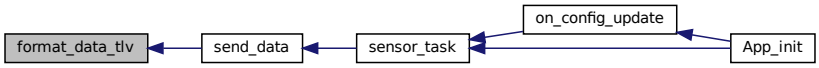
Size of the generated buffer. Or -1 if the buffer is to small.

Definition at line 25 of file format_data.c.

```
26 {
27     tlv_res_e tlv_ret = TLV_RES_ERROR;
28     tlv_record record;
29
30     const app_config_t * app_cfg = App_Config_get();
31
32     Tlv_init(&record, buffer, length);
33
34     /* Always add counter to packet. */
35     tlv_ret = Tlv_Encode_addItem(&record, TLV_TYPE_COUNTER,
36                                          sizeof(data->count),
37                                          &data->count);
38
39     if (app_cfg->temperature_enable && tlv_ret == TLV_RES_OK)
40     {
41         tlv_ret = Tlv_Encode_addItem(&record, TLV_TYPE_TEMPERATURE,
42                                          sizeof(data->temp),
43                                          &data->temp);
44     }
45
46     if (app_cfg->humidity_enable && tlv_ret == TLV_RES_OK)
47     {
48         tlv_ret = Tlv_Encode_addItem(&record, TLV_TYPE_HUMIDITY,
49                                          sizeof(data->humi),
50                                          &data->humi);
51     }
52
53     if (app_cfg->pressure_enable && tlv_ret == TLV_RES_OK)
54     {
55         tlv_ret = Tlv_Encode_addItem(&record, TLV_TYPE_PRESSURE,
56                                          sizeof(data->press),
57                                          &data->press);
58     }
59
60     if (app_cfg->accel_x_enable && tlv_ret == TLV_RES_OK)
61     {
62         tlv_ret = Tlv_Encode_addItem(&record, TLV_TYPE_ACCEL_X,
63                                          sizeof(data->acc_x),
64                                          &data->acc_x);
65     }
66
67     if (app_cfg->accel_y_enable && tlv_ret == TLV_RES_OK)
68     {
69         tlv_ret = Tlv_Encode_addItem(&record, TLV_TYPE_ACCEL_Y,
70                                          sizeof(data->acc_y),
71                                          &data->acc_y);
72     }
73
74     if (app_cfg->accel_z_enable && tlv_ret == TLV_RES_OK)
75     {
76         tlv_ret = Tlv_Encode_addItem(&record, TLV_TYPE_ACCEL_Z,
77                                          sizeof(data->acc_z),
78                                          &data->acc_z);
79     }
80
81     if (tlv_ret == TLV_RES_OK)
82     {
83         return Tlv_Encode_getBufferSize(&record);
84     }
85
86     return -1;
87 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



5.5 include/app_config.h File Reference

This module manages the application configuration.

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

Data Structures

- struct app_config_t
  Structure containing the application configuration.

Typedefs

- typedef void(∗ on_config_change_cb_f) (void)
  Function type config change callback.

Functions

- void App_Config_init (on_config_change_cb_f cb)
  Initialize the App Config module.
- const app_config_t ∗ App_Config_get (void)
  Returns a pointer to the configuration structure.

Detailed Description

This module manages the application configuration.

Copyright

Wirepas Oy 2019

Typedef Documentation

on_config_change_cb_f

```
typedef void(* on_config_change_cb_f) (void)
```

Function type config change callback.

Definition at line 30 of file app_config.h.

Function Documentation

App_Config_get()

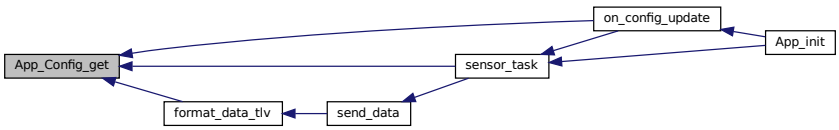
```
const app_config_t* App_Config_get (
                void  )
```

Returns a pointer to the configuration structure.

Definition at line 182 of file app_config.c.

```
183 {
184     return &m_config;
185 }
```

Here is the caller graph for this function:



App_Config_init()

```
void App_Config_init (
                on_config_change_cb_f cb )
```

Initialize the App Config module.

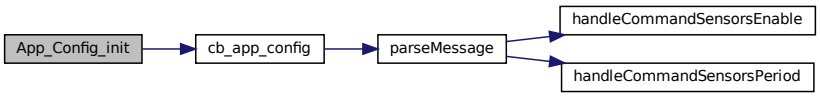
Parameters

in	*cb*	Pointer to a callback function. NULL if not used.

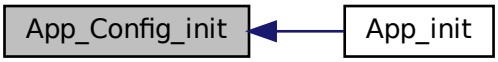
Definition at line 167 of file app_config.c.

```
168 {
169     m_config.temperature_enable =
    DEFAULT_TEMPERATURE_EN;
170     m_config.humidity_enable = DEFAULT_HUMIDITY_EN;
171     m_config.pressure_enable = DEFAULT_PRESSURE_EN;
172     m_config.accel_x_enable = DEFAULT_ACCEL_X_EN;
173     m_config.accel_y_enable = DEFAULT_ACCEL_Y_EN;
174     m_config.accel_z_enable = DEFAULT_ACCEL_Z_EN;
175     m_config.sensors_period_ms =
    DEFAULT_SENSORS_PERIOD_MS;
176
177     m_callback = cb;
178
179     lib_data->setNewAppConfigCb(cb_app_config);
180 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.6 include/bme280_wrapper.h File Reference

Wrapper on top of BME280 driver from Bosh GitHub project.

Data Structures

- struct bme280_wrapper_measurement_t
  Structure containing sensor measurements.

Functions

- bool BME280_wrapper_init (void)
  Initialize the BME280 wrapper library.
- uint32_t BME280_wrapper_startMeasurement (void)
  Start a measurement.
- bool BME280_wrapper_readMeasurement (bme280_wrapper_measurement_t ∗measurement)
  Get a measurement previously asked by a start measurement.

Detailed Description

Wrapper on top of BME280 driver from Bosh GitHub project.

This file is the wrapper to expose a simple API for getting the temperature, pressure and humidity out of device for a "weather application" configuration. It makes the glue between the Bosh driver and Wirepas SPI driver and staticaly configure the sensor in a given mode.

Copyright

Wirepas Oy 2019

Function Documentation

BME280_wrapper_init()

```
bool BME280_wrapper_init (
                void  )
```

Initialize the BME280 wrapper library.

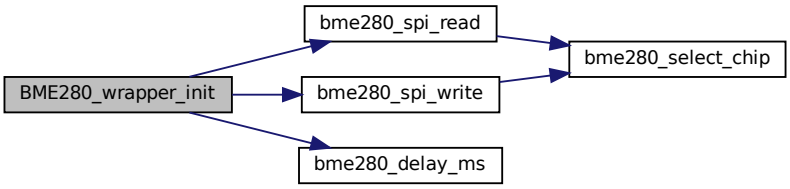
Returns

True if successfully initialized, false otherwise.

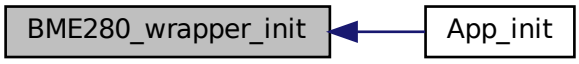
Definition at line 144 of file bme280_wrapper.c.

```
145 {
146     uint8_t settings_sel;
147
148     /* Create device for BME280 driver with the adapted read/write functions. */
149     m_bme280_dev.dev_id = 0;
150     m_bme280_dev.intf = BME280_SPI_INTF;
151     m_bme280_dev.read = bme280_spi_read;
152     m_bme280_dev.write = bme280_spi_write;
153     m_bme280_dev.delay_ms = bme280_delay_ms;
154
155     if (bme280_init(&m_bme280_dev) != BME280_OK)
156     {
157         return false;
158     }
159
160     /*
161      * Configure the mode of operation. It has a "Weather measurement" config
162      * as described in RM.
163      */
164     m_bme280_dev.settings.osr_h = BME280_OVERSAMPLING_1X;
165     m_bme280_dev.settings.osr_p = BME280_OVERSAMPLING_1X;
166     m_bme280_dev.settings.osr_t = BME280_OVERSAMPLING_1X;
167     m_bme280_dev.settings.filter = BME280_FILTER_COEFF_OFF;
168     settings_sel = BME280_OSR_PRESS_SEL |
169                    BME280_OSR_TEMP_SEL |
170                    BME280_OSR_HUM_SEL |
171                    BME280_FILTER_SEL;
172
173     if (bme280_set_sensor_settings(settings_sel, &m_bme280_dev) != BME280_OK)
174     {
175         return false;
176     }
177
178     return true;
179 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



BME280_wrapper_readMeasurement()

```
bool BME280_wrapper_readMeasurement (
                bme280_wrapper_measurement_t * measurement )
```

Get a measurement previously asked by a start measurement.

Parameters

in	*measurement*	The measurement read out from the sensor.

Returns

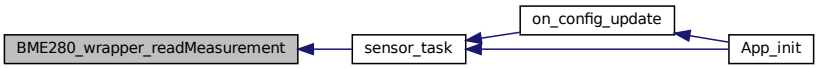
false if a problem occured, true otherwise.

Definition at line 196 of file bme280_wrapper.c.

```
197 {
198     struct bme280_data comp_data;
199     int8_t rslt;
200
201     rslt = bme280_get_sensor_data(BME280_ALL, &comp_data, &m_bme280_dev);
202     if (rslt != BME280_OK)
203     {
204         return false;
205     }
206
```

```
207     measurement->humidity = comp_data.humidity;
208     measurement->pressure = comp_data.pressure;
209     measurement->temperature = comp_data.temperature;
210
211     return true;
212 }
```

Here is the caller graph for this function:



BME280_wrapper_startMeasurement()

```
uint32_t BME280_wrapper_startMeasurement (
               void  )
```

Start a measurement.

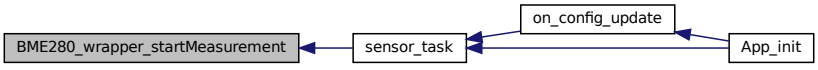
Returns

The time in ms to wait before the measurement is ready.

Definition at line 181 of file bme280_wrapper.c.

```
182 {
183     int8_t rslt;
184
185     /* Start a measurement. */
186     rslt = bme280_set_sensor_mode(BME280_FORCED_MODE, &m_bme280_dev);
187     (void) rslt;
188
189     /*
190      * According to our config, measurement max time is 9.3ms. Time can be
191      * computed according to chapter 9.3 of BME280 RM.
192      */
193     return BME280_MEASUREMENT_TIME_MS;
194 }
```

Here is the caller graph for this function:



5.7 include/format_data.h File Reference

This module contains functions to format sensor data to different format before sending them.

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

Data Structures

- struct sensor_data_t
  This structure contains all the sensors data to be sent.

Functions

- int format_data_tlv (uint8_t *buffer, sensor_data_t *data, int len)
  Format sensor data to TLV format.

Detailed Description

This module contains functions to format sensor data to different format before sending them.

Copyright

Wirepas Oy 2019

Function Documentation

format_data_tlv()

```
int format_data_tlv (
                uint8_t * buffer,
                sensor_data_t * data,
                int len )
```

Format sensor data to TLV format.

Formatted buffer looks like this: [Type] [Length] [Value] [0x01: Counter] [0x02] [uint16_t counter] [0x02: Temperature] [0x04] [int32_t temperature] [0x03: Humidity] [0x04] [uint32_t humidity] [0x04: Pressure] [0x04] [uint32_t pressure] [0x05: Accel X] [0x04] [int32_t accel X] [0x06: Accel Y] [0x04] [int32_t accel Y] [0x07: Accel Z] [0x04] [int32_t accel Z] Sensor data is only present in the formatted packet if it as been enabled in the configuration. Counter is always present.

Parameters

out	*buffer*	Buffer to store formatted data.
in	*data*	Pointer to the structure holding sensors data.
in	*len*	Length of the buffer

Returns

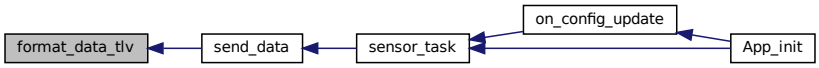
Size of the generated buffer. Or -1 if the buffer is to small.

Definition at line 25 of file format_data.c.

```
26 {
27     tlv_res_e tlv_ret = TLV_RES_ERROR;
28     tlv_record record;
29
30     const app_config_t * app_cfg = App_Config_get();
31
32     Tlv_init(&record, buffer, length);
33
34     /* Always add counter to packet. */
35     tlv_ret = Tlv_Encode_addItem(&record, TLV_TYPE_COUNTER,
36                                          sizeof(data->count),
37                                          &data->count);
38
39     if (app_cfg->temperature_enable && tlv_ret == TLV_RES_OK)
40     {
41         tlv_ret = Tlv_Encode_addItem(&record, TLV_TYPE_TEMPERATURE,
42                                          sizeof(data->temp),
43                                          &data->temp);
44     }
45
46     if (app_cfg->humidity_enable && tlv_ret == TLV_RES_OK)
47     {
48         tlv_ret = Tlv_Encode_addItem(&record, TLV_TYPE_HUMIDITY,
49                                          sizeof(data->humi),
50                                          &data->humi);
51     }
52
53     if (app_cfg->pressure_enable && tlv_ret == TLV_RES_OK)
54     {
55         tlv_ret = Tlv_Encode_addItem(&record, TLV_TYPE_PRESSURE,
56                                          sizeof(data->press),
57                                          &data->press);
58     }
59
60     if (app_cfg->accel_x_enable && tlv_ret == TLV_RES_OK)
61     {
62         tlv_ret = Tlv_Encode_addItem(&record, TLV_TYPE_ACCEL_X,
63                                          sizeof(data->acc_x),
64                                          &data->acc_x);
65     }
66
67     if (app_cfg->accel_y_enable && tlv_ret == TLV_RES_OK)
68     {
69         tlv_ret = Tlv_Encode_addItem(&record, TLV_TYPE_ACCEL_Y,
70                                          sizeof(data->acc_y),
71                                          &data->acc_y);
72     }
73
74     if (app_cfg->accel_z_enable && tlv_ret == TLV_RES_OK)
75     {
76         tlv_ret = Tlv_Encode_addItem(&record, TLV_TYPE_ACCEL_Z,
77                                          sizeof(data->acc_z),
78                                          &data->acc_z);
79     }
80
81     if (tlv_ret == TLV_RES_OK)
82     {
83         return Tlv_Encode_getBufferSize(&record);
84     }
85
86     return -1;
87 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



5.8 include/lis2dh12_wrapper.h File Reference

Wrapper on top of LIS2DH12 driver from STMicroelectronics GitHub.

Data Structures

- struct lis2dh12_wrapper_measurement_t
  Structure containing acceleration measurements.

Functions

- bool LIS2DH12_wrapper_init (void)
  Initialize the LIS2DH12 wrapper library.
- uint32_t LIS2DH12_wrapper_startMeasurement (void)
  Start a measurement.
- bool LIS2DH12_wrapper_readMeasurement (lis2dh12_wrapper_measurement_t *measurement)
  Get a measurement previously asked by a start measurement.

Detailed Description

Wrapper on top of LIS2DH12 driver from STMicroelectronics GitHub.

This file is the wrapper to expose a simple API for getting the acceleration from lis2dh12 device. It makes the glue between the ST driver and Wirepas SPI driver and staticaly configure the sensor.

Copyright

Wirepas Oy 2019

Function Documentation

LIS2DH12_wrapper_init()

```
bool LIS2DH12_wrapper_init (
                void  )
```

Initialize the LIS2DH12 wrapper library.

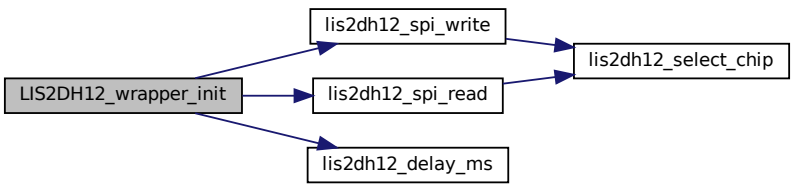
Returns

True if successfully initialized, false otherwise.

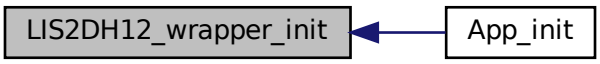
Definition at line 146 of file lis2dh12_wrapper.c.

```
147 {
148     uint8_t whoamI;
149 
150     /*
151      * Create device for LIS2DH12 driver with the adapted read/write functions.
152      */
153     m_lis2dh12_dev.write_reg = lis2dh12_spi_write;
154     m_lis2dh12_dev.read_reg = lis2dh12_spi_read;
155     m_lis2dh12_dev.handle = NULL;
156 
157     /* Set to Power Down. */
158     lis2dh12_data_rate_set(&m_lis2dh12_dev, LIS2DH12_POWER_DOWN);
159     lis2dh12_delay_ms(10);
160 
161     /* Check the device ID. */
162     lis2dh12_device_id_get(&m_lis2dh12_dev, &whoamI);
163     if (whoamI != LIS2DH12_ID)
164     {
165         /* Device not found or did not respond. */
166         return false;
167     }
168 
169     /* Reload memory. */
170     lis2dh12_boot_set(&m_lis2dh12_dev, 1);
171     lis2dh12_delay_ms(10);
172 
173     /* Enable Block Data Update. */
174     lis2dh12_block_data_update_set(&m_lis2dh12_dev, PROPERTY_ENABLE);
175 
176     /* Set full scale to 2g. */
177     lis2dh12_full_scale_set(&m_lis2dh12_dev, LIS2DH12_2g);
178 
179     /* Set device resolution to 12 bits. */
180     lis2dh12_operating_mode_set(&m_lis2dh12_dev, LIS2DH12_HR_12bit);
181 
182     return true;
183 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



LIS2DH12_wrapper_readMeasurement()

```
bool LIS2DH12_wrapper_readMeasurement (
              lis2dh12_wrapper_measurement_t * measurement )
```

Get a measurement previously asked by a start measurement.

Parameters

in	*measurement*	The measurement read out from the sensor.

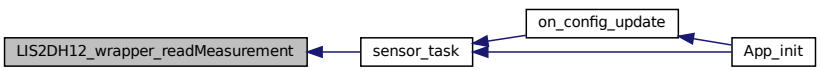
Returns

false if a problem occured, true otherwise.

Definition at line 196 of file lis2dh12_wrapper.c.

```
198 {
199     axis3bit16_t data_raw_acceleration;
200
201     /* Read accelerometer data. */
202     memset(data_raw_acceleration.u8bit, 0x00, 3*sizeof(int16_t));
203     lis2dh12_acceleration_raw_get(&m_lis2dh12_dev, data_raw_acceleration.u8bit);
204
205     /* Put the accelerometer back to sleep. */
206     lis2dh12_data_rate_set(&m_lis2dh12_dev, LIS2DH12_POWER_DOWN);
207
208     measurement->accel_x = data_raw_acceleration.i16bit[0] /
      RAW_ACCEL_TO_MG;
209     measurement->accel_y = data_raw_acceleration.i16bit[1] /
      RAW_ACCEL_TO_MG;
210     measurement->accel_z = data_raw_acceleration.i16bit[2] /
      RAW_ACCEL_TO_MG;
211
212     return true;
213 }
```

Here is the caller graph for this function:



LIS2DH12_wrapper_startMeasurement()

```
uint32_t LIS2DH12_wrapper_startMeasurement (
              void  )
```

Start a measurement.

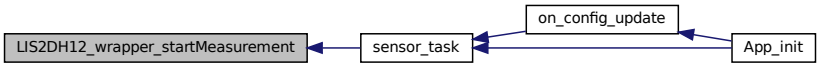
Returns

The time in ms to wait before the measurement is ready.

Definition at line 185 of file lis2dh12_wrapper.c.

```
186 {
187     lis2dh12_data_rate_set(&m_lis2dh12_dev, LIS2DH12_ODR_1Hz);
188
189     /*
190      * According to datasheet, wake up time from power down to high resolution
191      * @1Hz is 7ms.
192      */
193     return LIS2DH12_WAKE_UP_TIME_MS;
194 }
```

Here is the caller graph for this function:



5.9 include/main_page.h File Reference

5.10 lis2dh12_wrapper.c File Reference

Wrapper on top of LIS2DH12 driver from STMicroelectronics GitHub.

```
#include "lis2dh12_reg.h"
#include "board.h"
#include "hal_api.h"
#include "spi.h"
#include "api.h"
#include <string.h>
#include "lis2dh12_wrapper.h"
```

Macros

- #define LIS2DH12_WAKE_UP_TIME_MS (7∗1.5)
- #define RAW_ACCEL_TO_MG 16
- #define MAX_WRITE_SIZE 16
- #define MAX_READ_SIZE 16

Functions

- static void lis2dh12_delay_ms (uint32_t period)
  Blocking wait for a given amount of time.
- void lis2dh12_select_chip (bool select)
  Select or unselect LIS2DH12 with its chip select signal.
- static int32_t lis2dh12_spi_read (void ∗handle, uint8_t reg, uint8_t ∗bufp, uint16_t len)
  Read from SPI (function required by STMicroelectronics lib).
- static int32_t lis2dh12_spi_write (void ∗handle, uint8_t reg, uint8_t ∗bufp, uint16_t len)
  Write with SPI (function required by STMicroelectronics lib).
- bool LIS2DH12_wrapper_init (void)
  Initialize the LIS2DH12 wrapper library.
- uint32_t LIS2DH12_wrapper_startMeasurement (void)
  Start a measurement.
- bool LIS2DH12_wrapper_readMeasurement (lis2dh12_wrapper_measurement_t ∗measurement)
  Get a measurement previously asked by a start measurement.

Variables

- static lis2dh12_ctx_t m_lis2dh12_dev

Detailed Description

Wrapper on top of LIS2DH12 driver from STMicroelectronics GitHub.

Copyright

Wirepas Oy 2019

Macro Definition Documentation

LIS2DH12_WAKE_UP_TIME_MS

```
#define LIS2DH12_WAKE_UP_TIME_MS (7∗1.5)
```

Definition at line 15 of file lis2dh12_wrapper.c.

MAX_READ_SIZE

```
#define MAX_READ_SIZE 16
```

Definition at line 24 of file lis2dh12_wrapper.c.

MAX_WRITE_SIZE

```
#define MAX_WRITE_SIZE 16
```

Definition at line 21 of file lis2dh12_wrapper.c.

RAW_ACCEL_TO_MG

```
#define RAW_ACCEL_TO_MG 16
```

Definition at line 18 of file lis2dh12_wrapper.c.

Function Documentation

lis2dh12_delay_ms()

```
static void lis2dh12_delay_ms (
               uint32_t period )  [static]
```

Blocking wait for a given amount of time.

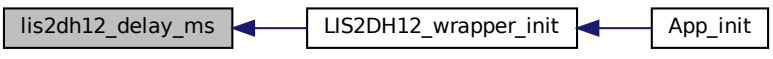
Parameters

in	*period*	Time to wait in ms.

Definition at line 33 of file lis2dh12_wrapper.c.

```
34 {
35     app_lib_time_timestamp_hp_t end;
36     end = lib_time->addUsToHpTimestamp(lib_time->getTimestampHp(),
37                                        period * 1000);
38
39     /* Active wait until period is elapsed */
40     while (lib_time->isHpTimestampBefore(lib_time->getTimestampHp(),
41                                          end));
42 }
```

Here is the caller graph for this function:



lis2dh12_select_chip()

```
void lis2dh12_select_chip (
                bool select )
```

Select or unselect LIS2DH12 with its chip select signal.

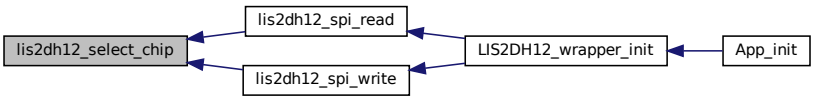
Parameters

in	*select*	True to select and false to unselect.

Definition at line 48 of file lis2dh12_wrapper.c.

```
49 {
50     if (select)
51     {
52         nrf_gpio_pin_clear(BOARD_SPI_LIS2DH12_CS_PIN);
53     }
54     else
55     {
56         nrf_gpio_pin_set(BOARD_SPI_LIS2DH12_CS_PIN);
57     }
58 }
```

Here is the caller graph for this function:



lis2dh12_spi_read()

```
static int32_t lis2dh12_spi_read (
                void * handle,
                uint8_t reg,
                uint8_t * bufp,
                uint16_t len )  [static]
```

Read from SPI (function required by STMicroelectronics lib).

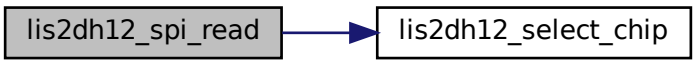
Parameters

in	*handle*	SPI driver id (unused here).
in	*reg*	First register to read.
in	*bufp*	Pointer to store read registers.
in	*len*	Number of registers (of 1 byte) to read.

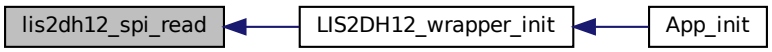
Definition at line 67 of file lis2dh12_wrapper.c.

```
71 {
72     spi_res_e res;
73     uint8_t tx[1];
74     uint8_t rx[MAX_READ_SIZE + 1];
75 
76     if (len > MAX_READ_SIZE)
77     {
78         return -1;
79     }
80 
81     tx[0] = reg | 0xC0;
82 
83     spi_xfer_t transfer;
84     transfer.write_ptr = tx;
85     transfer.write_size = 1;
86     transfer.read_ptr = rx;
87     transfer.read_size = len + 1;
88 
89     lis2dh12_select_chip(true);
90     /* Blocking read. */
91     res = SPI_transfer(&transfer, NULL);
92     lis2dh12_select_chip(false);
93 
94     if (res != SPI_RES_OK)
95     {
96         return -1;
97     }
98 
99     memcpy(bufp, &rx[1], len);
100 
101     return 0;
102 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



lis2dh12_spi_write()

```
static int32_t lis2dh12_spi_write (
               void * handle,
               uint8_t reg,
               uint8_t * bufp,
               uint16_t len )  [static]
```

Write with SPI (function required by STMicroelectronics lib).

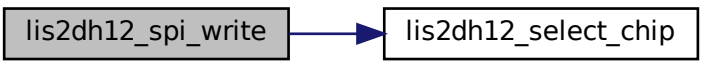
Parameters

in	*handle*	SPI driver id (unused here).
in	*reg*	First register to write to.
out	*bufp*	Pointer in RAM to the data to be written.
in	*len*	Number of registers (of 1 byte) to write.

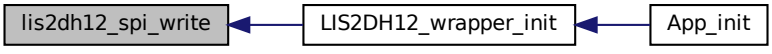
Definition at line 111 of file lis2dh12_wrapper.c.

```
115 {
116     spi_res_e res;
117     uint8_t tx[MAX_WRITE_SIZE + 1];
118 
119     if (len > MAX_WRITE_SIZE)
120     {
121         return -1;
122     }
123 
124     tx[0] = reg | 0x40;
125     memcpy(&tx[1], bufp, len);
126 
127     spi_xfer_t transfer;
128     transfer.write_ptr = tx;
129     transfer.write_size = len + 1;
130     transfer.read_ptr = NULL;
131     transfer.read_size = 0;
132 
133     lis2dh12_select_chip(true);
134     /* Blocking write */
135     res = SPI_transfer(&transfer, NULL);
136     lis2dh12_select_chip(false);
137 
138     if (res != SPI_RES_OK)
139     {
140         return -1;
141     }
142 
143     return 0;
144 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



LIS2DH12_wrapper_init()

```
bool LIS2DH12_wrapper_init (
                void  )
```

Initialize the LIS2DH12 wrapper library.

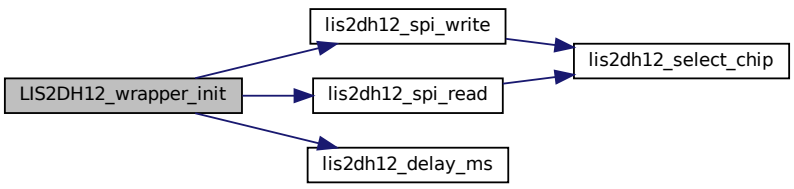
Returns

True if successfully initialized, false otherwise.

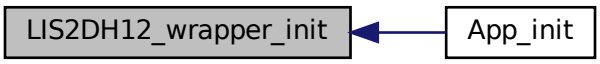
Definition at line 146 of file lis2dh12_wrapper.c.

```
147 {
148     uint8_t whoamI;
149 
150     /*
151      * Create device for LIS2DH12 driver with the adapted read/write functions.
152      */
153     m_lis2dh12_dev.write_reg = lis2dh12_spi_write;
154     m_lis2dh12_dev.read_reg = lis2dh12_spi_read;
155     m_lis2dh12_dev.handle = NULL;
156 
157     /* Set to Power Down. */
158     lis2dh12_data_rate_set(&m_lis2dh12_dev, LIS2DH12_POWER_DOWN);
159     lis2dh12_delay_ms(10);
160 
161     /* Check the device ID. */
162     lis2dh12_device_id_get(&m_lis2dh12_dev, &whoamI);
163     if (whoamI != LIS2DH12_ID)
164     {
165         /* Device not found or did not respond. */
166         return false;
167     }
168 
169     /* Reload memory. */
170     lis2dh12_boot_set(&m_lis2dh12_dev, 1);
171     lis2dh12_delay_ms(10);
172 
173     /* Enable Block Data Update. */
174     lis2dh12_block_data_update_set(&m_lis2dh12_dev, PROPERTY_ENABLE);
175 
176     /* Set full scale to 2g. */
177     lis2dh12_full_scale_set(&m_lis2dh12_dev, LIS2DH12_2g);
178 
179     /* Set device resolution to 12 bits. */
180     lis2dh12_operating_mode_set(&m_lis2dh12_dev, LIS2DH12_HR_12bit);
181 
182     return true;
183 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



LIS2DH12_wrapper_readMeasurement()

```
bool LIS2DH12_wrapper_readMeasurement (
              lis2dh12_wrapper_measurement_t * measurement )
```

Get a measurement previously asked by a start measurement.

Parameters

in	*measurement*	The measurement read out from the sensor.

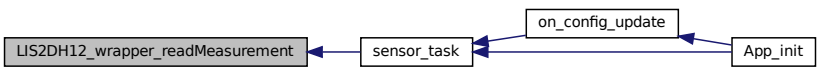
Returns

false if a problem occured, true otherwise.

Definition at line 196 of file lis2dh12_wrapper.c.

```
198 {
199     axis3bit16_t data_raw_acceleration;
200
201     /* Read accelerometer data. */
202     memset(data_raw_acceleration.u8bit, 0x00, 3*sizeof(int16_t));
203     lis2dh12_acceleration_raw_get(&m_lis2dh12_dev, data_raw_acceleration.u8bit);
204
205     /* Put the accelerometer back to sleep. */
206     lis2dh12_data_rate_set(&m_lis2dh12_dev, LIS2DH12_POWER_DOWN);
207
208     measurement->accel_x = data_raw_acceleration.i16bit[0] /
      RAW_ACCEL_TO_MG;
209     measurement->accel_y = data_raw_acceleration.i16bit[1] /
      RAW_ACCEL_TO_MG;
210     measurement->accel_z = data_raw_acceleration.i16bit[2] /
      RAW_ACCEL_TO_MG;
211
212     return true;
213 }
```

Here is the caller graph for this function:



LIS2DH12_wrapper_startMeasurement()

```
uint32_t LIS2DH12_wrapper_startMeasurement (
               void  )
```

Start a measurement.

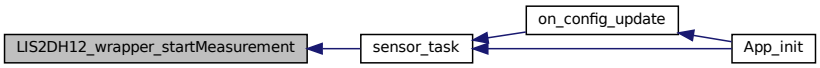
Returns

The time in ms to wait before the measurement is ready.

Definition at line 185 of file lis2dh12_wrapper.c.

```
186 {
187     lis2dh12_data_rate_set(&m_lis2dh12_dev, LIS2DH12_ODR_1Hz);
188
189     /*
190      * According to datasheet, wake up time from power down to high resolution
191      * @1Hz is 7ms.
192      */
193     return LIS2DH12_WAKE_UP_TIME_MS;
194 }
```

Here is the caller graph for this function:



Variable Documentation

m_lis2dh12_dev

```
lis2dh12_ctx_t m_lis2dh12_dev  [static]
```

Definition at line 27 of file lis2dh12_wrapper.c.

Index