Table of Contents

1 Introduction

1.1 Purpose

This document helps the BME680 user to integrate the sensor API/ Driver code to their Framework.

1.2 Scope

This document covers most possible use cases of the BME680 sensor and required configuration.

1.3 Definitions, Acronyms and Abbreviations

Table1: Terms and definitions

Term	Definition
API	Application Program interface
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
RAM	Random Access Memory
ROM	Read Only Memory
T+P+G+H	Temperature, Pressure, Humidity & Gas sensor
.bss	Block Started by Symbol

1.4 References

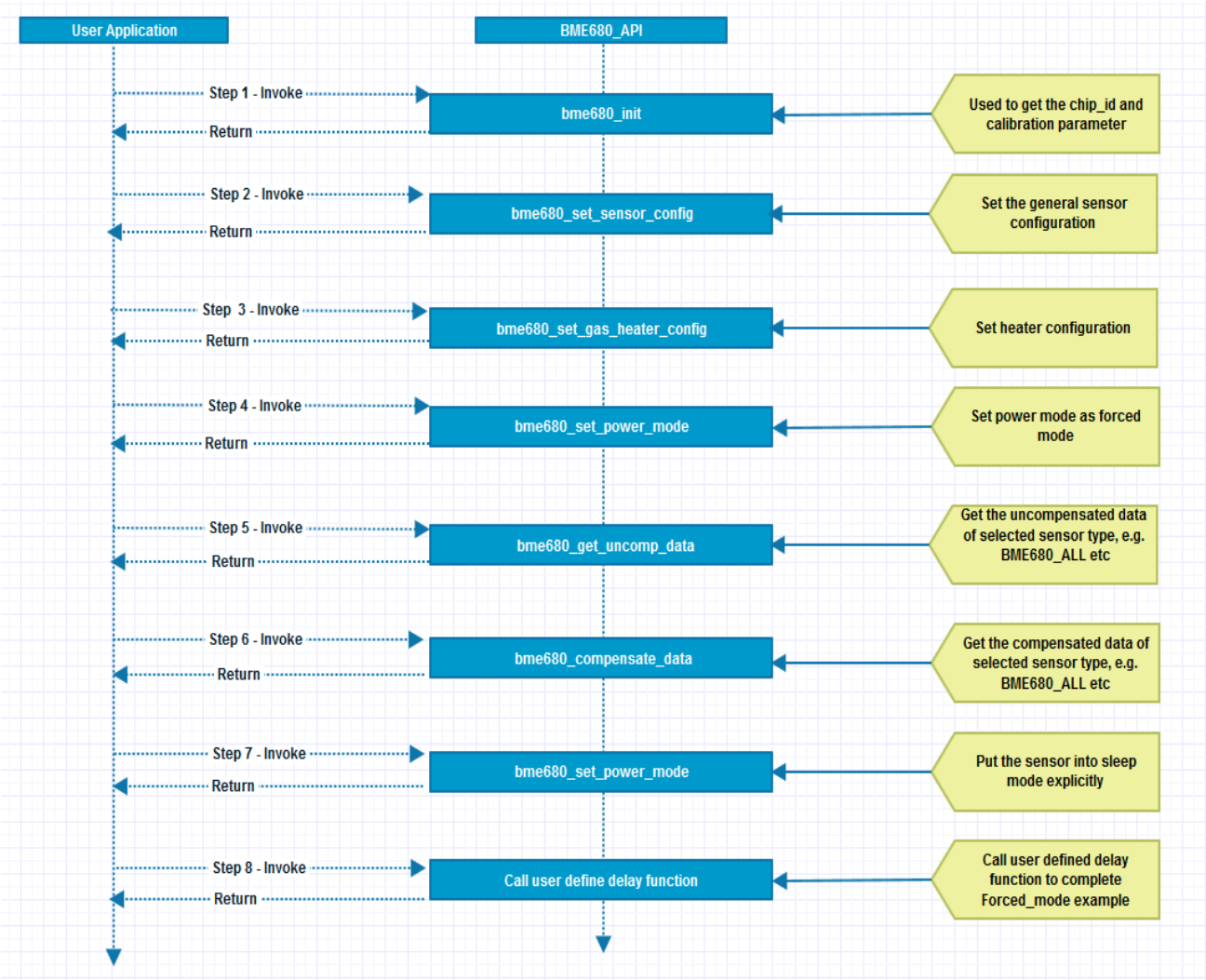
- BME680 Data sheet

2 Example use cases

2.1 BME680 single sensor – Force Mode – T+P+H+G Enabled

This example represent the use case of single BME680 sensor interfaced over SPI and reads the compensated data of Temperature, Pressure, Humidity & Gas sensor according to sensor settings in forced mode.

2.1.1 Sequence diagram



2.1.1 Example code

```
/* BME680 sensor – Force Mode – T+P+H+G Enabled */

/* include bme680 main header */
#include "bme680.h"
/*!
*       BME680_MAX_NO_OF_SENSOR = 2; defined in bme680.h file
*       In order to interface only one sensor over SPI, user must change the value of
*       BME680_MAX_NO_OF_SENSOR = 1
*       Test setup: It has been assumed that “BME680 sensor_0” interfaced over SPI with
*       Native chip select line
*/

/* BME680 sensor structure instance */
struct bme680_t bme680_sensor_no[BME680_MAX_NO_OF_SENSOR];

/* BME680 sensor’s compensated data structure instance */
struct bme680_comp_field_data compensate_data_sensor[BME680_MAX_NO_OF_SENSOR][3];

/* BME680 sensor’s uncompensated data structure instance */
struct bme680_uncomp_field_data uncompensated_data_of_sensor[BME680_MAX_NO_OF_SENSOR][3];

/* BME680 sensor’s configuration structure instance */
struct bme680_sens_conf set_conf_sensor[BME680_MAX_NO_OF_SENSOR];

/* BME680 sensor’s heater configuration structure instance */
struct bme680_heater_conf set_heatr_conf_sensor[BME680_MAX_NO_OF_SENSOR];

void main(void)
{

        unsigned int i = BME680_INIT_VALUE;

        enum bme680_return_type com_rslt = BME680_COMM_RES_ERROR;

        /* Do BME680 sensor structure instance initialization*/

        /* Sensor_0 interface over SPI with native chip select line */
        /* USER defined SPI bus read function       */
        bme680_sensor_no[0].bme680_bus_read = BME680_SPI_bus_read_user;
        /* USER defined SPI bus write function      */
        bme680_sensor_no[0].bme680_bus_write = BME680_SPI_bus_write_user;
        /* USER defined SPI burst read function */
        bme680_sensor_no[0].bme680_burst_read = BME680_SPI_bus_read_user;
        /* USER defined delay function */
        bme680_sensor_no[0].delay_msec = BME680_delay_msec_user;
        /* Mention communication interface */
        bme680_sensor_no[0].interface = BME680_SPI_INTERFACE;



        /* get chip id and calibration parameter */
        com_rslt = bme680_init(&bme680_sensor_no[0]);

        /* Do Sensor initialization */
        for (i=0;i<BME680_MAX_NO_OF_SENSOR;i++) {


        /* Check Device-ID before next steps of sensor operations */
        if (BME680_CHIP_ID == bme680_sensor_no[i].chip_id) {
```

```
        /* Select sensor configuration parameters */
                set_conf_sensor[i].heatr_ctrl = BME680_HEATR_CTRL_ENABLE;
                set_conf_sensor[i].run_gas = BME680_RUN_GAS_ENABLE;
                set_conf_sensor[i].nb_conv = 0x00;
                set_conf_sensor[i].osrs_hum = BME680_OSRS_1X;
                set_conf_sensor[i].osrs_pres = BME680_OSRS_1X;
                set_conf_sensor[i].osrs_temp = BME680_OSRS_1X;

                /* activate sensor configuration */
                com_rslt += bme680_set_sensor_config(&set_conf_sensor[i],
                                                     &bme680_sensor_no[i]);


                /* Select Heater configuration parameters */
                set_heatr_conf_sensor[i].heater_temp[0] = 300;
                set_heatr_conf_sensor[i].heatr_idacv[0] = 1;
                set_heatr_conf_sensor[i].heatr_dur[0] = 137;
                set_heatr_conf_sensor[i].profile_cnt = 1;

                /* activate heater configuration */
                com_rslt += bme680_set_gas_heater_config(&set_heatr_conf_sensor[i],
                                                       &bme680_sensor_no[i]);

                /* Set power mode as forced mode */
                com_rslt += bme680_set_power_mode(BME680_FORCED_MODE,&bme680_sensor_no[i]);

        if (BME680_COMM_RES_OK == com_rslt) {
                /*Get the uncompensated T+P+G+H data*/
                bme680_get_uncomp_data(uncompensated_data_of_sensor[i], 1, BME680_ALL,
                &bme680_sensor_no[i]);

                /*Get the compensated T+P+G+H data*/

                bme680_compensate_data(uncompensated_data_of_sensor[i],
                                       compensate_data_sensor[i], 1,
                                       BME680_ALL, &bme680_sensor_no[i]);

                /* put sensor into sleep mode explicitly */
                bme680_set_power_mode(BME680_SLEEP_MODE, &bme680_sensor_no[i]);

                /* call user define delay function(duration millisecond) */
                User_define_delay(100);

                }
          }
     }
}
```

NOTE : An user defined delay should be added in between setting the power mode (bme680_set_power_mode) and reading the data (bme680_get_uncomp_data) ,which is the sum of delays due to conversion of P, T, rH, Heater duration and gas resistance required for the proper data readout

2.2 BME680 multiple sensor – Force Mode – T+P+H+G Enabled

This example represent the use case of BME680 (two bme680 interfaced over I2C & SPI) in force mode operation. Read the Temperature, Pressure, Humidity & Gas sensor data according to sensor settings.

2.2.1 Sequence diagram

Below diagram gives the API calls used for this example and clearly shows the sequence of the function calls to be used.

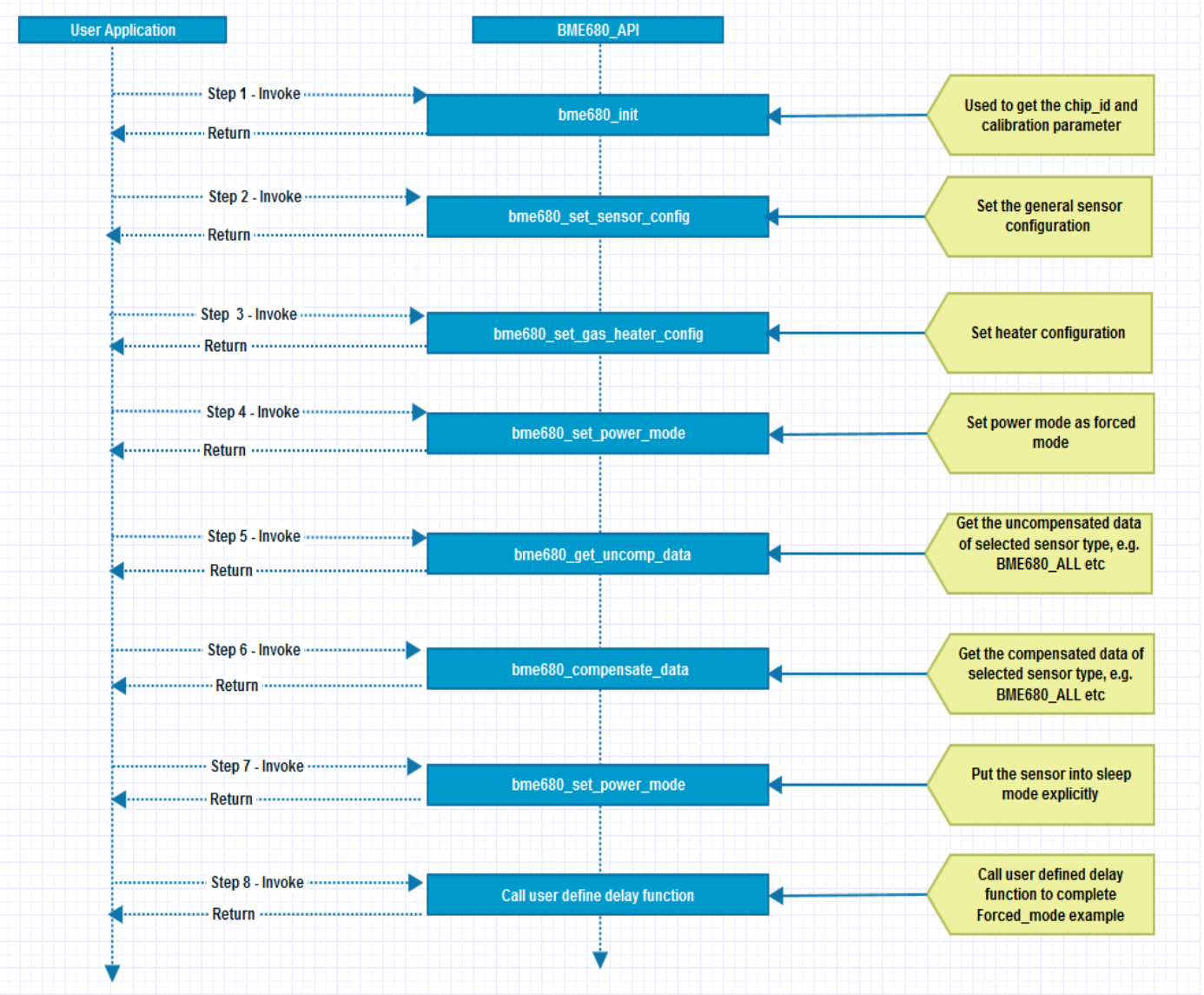


Figure 1 Forced mode

2.2.2 Example code

```
/* BME680 two sensor – Force Mode – T+P+H+G Enabled */

/* include bme680 main header */
#include "bme680.h"
/*!
*          BME680_MAX_NO_OF_SENSOR = 2; defined in bme680.h file
*          Note: user can change sensor number according to requirements.
*          Test setup: It has been assumed that “BME680 sensor_0” interfaced over SPI and
*          “BME680 sensor_1” over I2C with device address 0x77(SDO level = HIGH)
*/
 /* BME680 sensor structure instance */
struct bme680_t bme680_sensor_no[BME680_MAX_NO_OF_SENSOR];

/* BME680 sensor’s compensated data structure instance */
struct bme680_comp_field_data compensate_data_sensor[BME680_MAX_NO_OF_SENSOR][3];

/* BME680 sensor’s uncompensated data structure instance */
struct bme680_uncomp_field_data uncompensated_data_of_sensor[BME680_MAX_NO_OF_SENSOR][3];

/* BME680 sensor’s configuration structure instance */
struct bme680_sens_conf set_conf_sensor[BME680_MAX_NO_OF_SENSOR];

/* BME680 sensor’s heater configuration structure instance */
struct bme680_heater_conf set_heatr_conf_sensor[BME680_MAX_NO_OF_SENSOR];

void main(void)
{
        unsigned int i = BME680_INIT_VALUE;

        enum bme680_return_type com_rslt = BME680_COMM_RES_ERROR;

        /* Do BME680 sensor structure instance initialization*/

        /* Sensor_0 interface over SPI with native chip select line */
        /* USER defined SPI bus read function        */
        bme680_sensor_no[0].bme680_bus_read = BME680_SPI_bus_read_user;
        /* USER defined SPI bus write function       */
        bme680_sensor_no[0].bme680_bus_write = BME680_SPI_bus_write_user;
        /* USER defined SPI burst read  function */
        bme680_sensor_no[0].bme680_burst_read = BME680_SPI_bus_read_user;
        /* USER defined delay function */
        bme680_sensor_no[0].delay_msec = BME680_delay_msec_user;
        /* Mention communication interface */
        bme680_sensor_no[0].interface = BME680_SPI_INTERFACE;


        /* Sensor_1 interface over I2C with address as 0x77(SDO level = HIGH) */
        /* USER defined I2C bus write function       */
        bme680_sensor_no[1].bme680_bus_write = BME680_I2C_bus_write_user;
        /* USER defined I2C bus read function        */
        bme680_sensor_no[1].bme680_bus_read = BME680_I2C_bus_read_user;
        /* USER defined I2C burst read function */
        bme680_sensor_no[1].bme680_burst_read = BME680_I2C_bus_read_user;
        /* USER defined delay function       */
        bme680_sensor_no[1].delay_msec = BME680_delay_msec_user;
        /*set the address according to sensor's SDO level and interfaced protocol */
        bme680_sensor_no[1].dev_addr = BME680_I2C_ADDR_SECONDARY;
        /* Mention communication interface */
        bme680_sensor_no[1].interface = BME680_I2C_INTERFACE;

        /* get chip id and calibration parameter */
        com_rslt = bme680_init(&bme680_sensor_no[0]);
```

```
        com_rslt += bme680_init(&bme680_sensor_no[1]);

        /* Do Sensor initialization */
        for (i=0; I < BME680_MAX_NO_OF_SENSOR; i++) {

        /* Check Device-ID before next steps of sensor operations */
        if (BME680_CHIP_ID == bme680_sensor_no[i].chip_id) {

        /* Select sensor configuration parameters */
                set_conf_sensor[i].heatr_ctrl = BME680_HEATR_CTRL_ENABLE;
                set_conf_sensor[i].run_gas = BME680_RUN_GAS_ENABLE;
                set_conf_sensor[i].nb_conv = 0x00;
                set_conf_sensor[i].osrs_hum = BME680_OSRS_1X;
                set_conf_sensor[i].osrs_pres = BME680_OSRS_1X;
                set_conf_sensor[i].osrs_temp = BME680_OSRS_1X;

                /* activate sensor configuration */
                com_rslt += bme680_set_sensor_config(&set_conf_sensor[i], &bme680_sensor_no[i]);


                /* Select Heater configuration parameters */
                set_heatr_conf_sensor[i].heater_temp[0] = 300;
                set_heatr_conf_sensor[i].heatr_idacv[0] = 1;
                set_heatr_conf_sensor[i].heatr_dur[0] = 137;
                set_heatr_conf_sensor[i].profile_cnt = 1;

                /* activate heater configuration */
                com_rslt += bme680_set_gas_heater_config(&set_heatr_conf_sensor[i], &bme680_sensor_no[i]);

                /* Set power mode as forced mode */
                com_rslt += bme680_set_power_mode(BME680_FORCED_MODE,&bme680_sensor_no[i]);

        if (BME680_COMM_RES_OK == com_rslt) {
                /*Get the uncompensated T+P+G+H data*/
                bme680_get_uncomp_data(uncompensated_data_of_sensor[i], 1, BME680_ALL, &bme680_sen-
                sor_no[i]);

                /*Get the compensated T+P+G+H data*/
                bme680_compensate_data(uncompensated_data_of_sensor[i], compensate_data_sensor[i], 1,
                BME680_ALL, &bme680_sensor_no[i]);

                /* put sensor into sleep mode explicitly */
                bme680_set_power_mode(BME680_SLEEP_MODE, &bme680_sensor_no[i]);

                /* call user define delay function (duration millisecond) */
                User_define_delay(100);
                }
          }
      }
}
```

NOTE : An user defined delay should be added in between setting the power mode (bme680_set_power_mode) and reading the data (bme680_get_uncomp_data) ,which is the sum of delays due to conversion of P, T, rH, Heater duration and gas resistance required for the proper data readout

2.3 BME680 multiple sensor – sequential Mode – T+P+H+G Enabled

This example represent the use case of BME680 (two bme680 interfaced over I2C & SPI) in sequential mode operation and read compensated data of Temperature, Pressure, and Humidity & Gas sensor according to settings.

2.3.1 Sequence diagram

Below diagram gives the API calls used for this example and clearly shows the sequence of the function calls to be used.

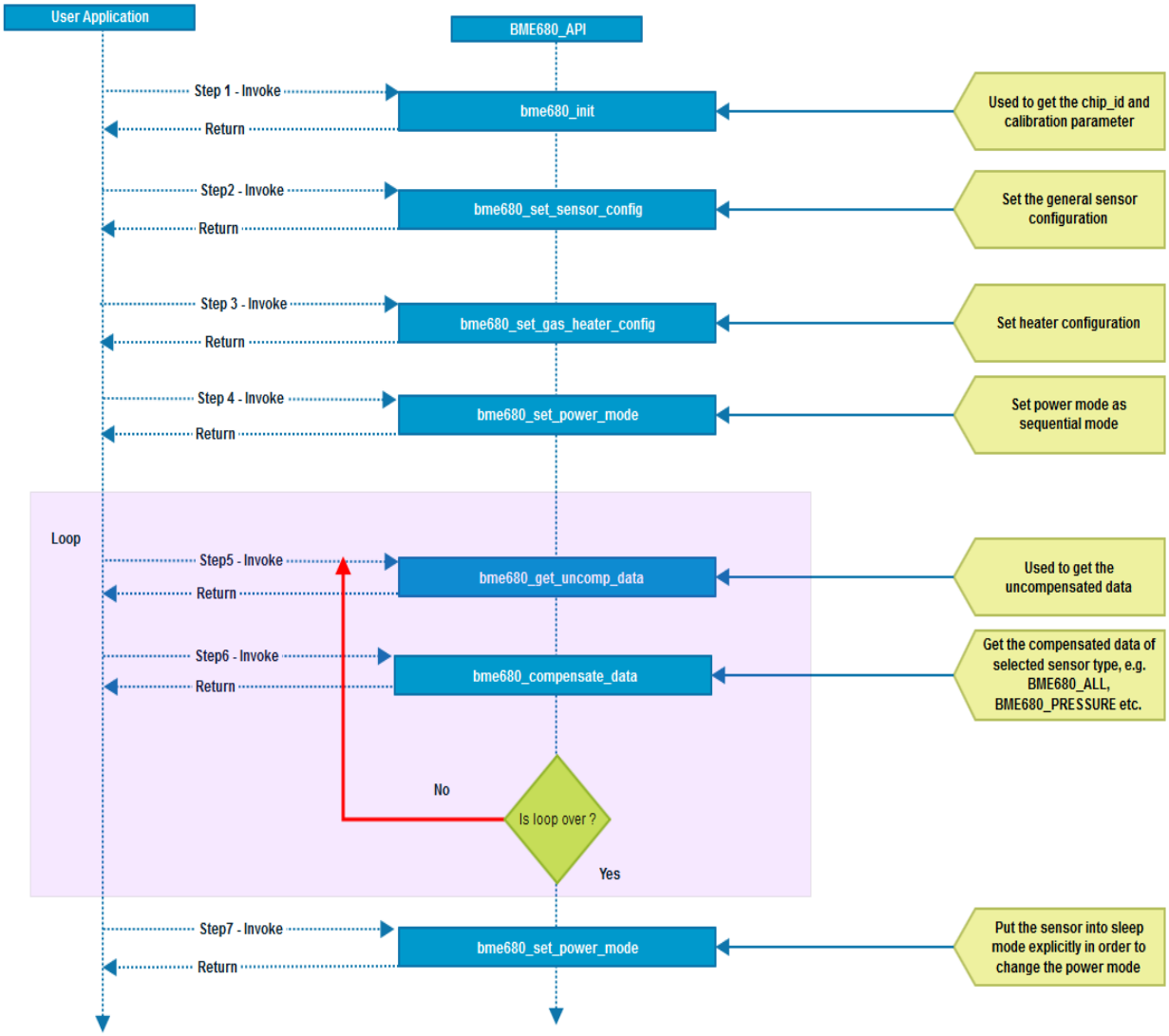


Figure 2 Sequential mode

2.3.2 Example code

```
/* BME680 two sensor – sequential Mode – T+P+H+G Enabled */

/* include bme680 main header */
#include "bme680.h"
/*!
*          BME680_MAX_NO_OF_SENSOR = 2; defined in bme680.h file
*          Note: user can change sensor number according to requirements.
*          Test setup: It has been assumed that “BME680 sensor_0” interfaced over SPI and
*          “BME680 sensor_1” over I2C with device address 0x77(SDO level = HIGH)
*/

#define READ_COUNT 10
/* BME680 sensor structure instance */
struct bme680_t bme680_sensor_no[BME680_MAX_NO_OF_SENSOR];

/* BME680 sensor’s compensated data structure instance */
struct bme680_comp_field_data compensate_data_sensor[BME680_MAX_NO_OF_SENSOR][3];

/* BME680 sensor’s uncompensated data structure instance */
struct bme680_uncomp_field_data uncompensated_data_of_sensor[BME680_MAX_NO_OF_SENSOR][3];

/* BME680 sensor’s configuration structure instance */
struct bme680_sens_conf set_conf_sensor[BME680_MAX_NO_OF_SENSOR];

/* BME680 sensor’s heater configuration structure instance */
struct bme680_heater_conf set_heatr_conf_sensor[BME680_MAX_NO_OF_SENSOR];

void main(void)
{

        unsigned int i = BME680_INIT_VALUE;

        int j = BME680_INT_VALUE;

        enum bme680_return_type com_rslt = BME680_COMM_RES_ERROR;

        /* Do BME680 sensor structure instance initialization*/

        /* Sensor_0 interface over SPI with native chip select line */

        /* USER defined SPI bus read function        */
        bme680_sensor_no[0].bme680_bus_read = BME680_SPI_bus_read_user;
        /* USER defined SPI bus write function       */
        bme680_sensor_no[0].bme680_bus_write = BME680_SPI_bus_write_user;
        /* USER defined SPI burst read  function */
        bme680_sensor_no[0].bme680_burst_read = BME680_SPI_bus_read_user;
        /* USER defined delay function */
        bme680_sensor_no[0].delay_msec = BME680_delay_msec_user;
        /* Mention communication interface */
        bme680_sensor_no[0].interface = BME680_SPI_INTERFACE;


        /* Sensor_1 interface over I2C with address as 0x77(SDO level = HIGH) */

        /* USER defined I2C bus write function       */
        bme680_sensor_no[1].bme680_bus_write = BME680_I2C_bus_write_user;
        /* USER defined I2C bus read function        */
        bme680_sensor_no[1].bme680_bus_read = BME680_I2C_bus_read_user;
        /* USER defined I2C burst read function */
        bme680_sensor_no[1].bme680_burst_read = BME680_I2C_bus_read_user;
        /* USER defined delay function       */
        bme680_sensor_no[1].delay_msec = BME680_delay_msec_user;
```

```
/*set the address according to sensor's SDO level and interfaced protocol */
bme680_sensor_no[1].dev_addr = BME680_I2C_ADDR_SECONDARY;
/* Mention communication interface */
bme680_sensor_no[1].interface = BME680_I2C_INTERFACE;

/* get chip id and calibration parameter */
com_rslt = bme680_init(&bme680_sensor_no[0]);
com_rslt += bme680_init(&bme680_sensor_no[1]);

/* Do Sensor initialization */
for (i=0; i < BME680_MAX_NO_OF_SENSOR; i++) {


/* Check Device-ID before next steps of sensor operations */
if (BME680_CHIP_ID == bme680_sensor_no[i].chip_id) {

/* Select sensor configuration parameters */

set_conf_sensor[i].heatr_ctrl = BME680_HEATR_CTRL_ENABLE;
set_conf_sensor[i].odr = BME680_ODR_20MS;
set_conf_sensor[i].run_gas = BME680_RUN_GAS_DISABLE;
set_conf_sensor[i].nb_conv = 0x09;
set_conf_sensor[i].osrs_hum = BME680_OSRS_1X;
set_conf_sensor[i].osrs_pres = BME680_OSRS_2X;
set_conf_sensor[i].osrs_temp = BME680_OSRS_4X;
set_conf_sensor[i].filter = BME680_FILTER_COEFF_0;

/* activate sensor configuration */
com_rslt += bme680_set_sensor_config(&set_conf_sensor[i],
                                     &bme680_sensor_no[i]);


/* Select Heater configuration parameters */
set_heatr_conf_sensor[i].heatr_dur[0] = 0;
set_heatr_conf_sensor[i].heatr_dur[1] = 65536;
set_heatr_conf_sensor[i].heatr_dur[2] = 89;
set_heatr_conf_sensor[i].heatr_dur[3] = 90;
set_heatr_conf_sensor[i].heatr_dur[4] = 91;
set_heatr_conf_sensor[i].heatr_dur[5] = 92;
set_heatr_conf_sensor[i].heatr_dur[6] = 93;
set_heatr_conf_sensor[i].heatr_dur[7] = 94;
set_heatr_conf_sensor[i].heatr_dur[8] = 95;
set_heatr_conf_sensor[i].heatr_dur[9] = 96;

set_heatr_conf_sensor[i].heatr_idacv[0] = 1;
set_heatr_conf_sensor[i].heatr_idacv[1] = 2;
set_heatr_conf_sensor[i].heatr_idacv[2] = 3;
set_heatr_conf_sensor[i].heatr_idacv[3] = 4;
set_heatr_conf_sensor[i].heatr_idacv[4] = 5;
set_heatr_conf_sensor[i].heatr_idacv[5] = 6;
set_heatr_conf_sensor[i].heatr_idacv[6] = 7;
set_heatr_conf_sensor[i].heatr_idacv[7] = 8;
set_heatr_conf_sensor[i].heatr_idacv[8] = 9;
set_heatr_conf_sensor[i].heatr_idacv[9] = 10;

set_heatr_conf_sensor[i].heater_temp[0] = 199;
set_heatr_conf_sensor[i].heater_temp[1] = 200;
set_heatr_conf_sensor[i].heater_temp[2] = 201;
set_heatr_conf_sensor[i].heater_temp[3] = 247;
set_heatr_conf_sensor[i].heater_temp[4] = 248;
set_heatr_conf_sensor[i].heater_temp[5] = 249;
set_heatr_conf_sensor[i].heater_temp[6] = 250;
set_heatr_conf_sensor[i].heater_temp[7] = 399;
set_heatr_conf_sensor[i].heater_temp[8] = 401;
```

```
        set_heatr_conf_sensor[i].heater_temp[9] = 400;

        set_heatr_conf_sensor[i].heatr_dur_shared = 200;
        set_heatr_conf_sensor[i].profile_cnt = 10;

        /* activate heater configuration */
        com_rslt += bme680_set_gas_heater_config(&set_heatr_conf_sensor[i],
        &bme680_sensor_no[i]);

        /* Set power mode as sequential mode */
        com_rslt += bme680_set_power_mode(BME680_SEQUENTIAL_MODE,&bme680_sensor_no[i]);

        if (BME680_COMM_RES_OK == com_rslt) {

                for (j = 0; j < READ_COUNT; j++) {

                /*Get the uncompensated T+P+G+H data*/
                bme680_get_uncomp_data(uncompensated_data_of_sensor[i], 3,
                                        BME680_ALL, &bme680_sensor_no[i]);

                /*Get the compensated T+P+G+H data*/
                bme680_compensate_data(uncompensated_data_of_sensor[i],
                                        compensate_data_sensor[i], 3,
                                        BME680_ALL,
                                        &bme680_sensor_no[i]);
                }

                /* put sensor into sleep mode explicitly */
        bme680_set_power_mode(BME680_SLEEP_MODE, &bme680_sensor_no[i]);

         }
      }
   }
}
```

NOTE : An user defined delay should be added in between setting the power mode (bme680_set_power_mode) and reading the data (bme680_get_uncomp_data) ,which is the sum of delays due to conversion of P, T, rH, Heater duration and gas resistance required for the proper data readout

2.4 BME680 multiple sensor – parallel Mode – T+P+H+G Enabled

This example represent the use case of BME680 (two bme680 interfaced over I2C & SPI) in parallel mode operation and read compensated data of Temperature, Pressure, Humidity & Gas sensor according to settings

2.4.1 Sequence diagram

Below diagram gives the API calls used for this example and clearly shows the sequence of the function calls to be used.

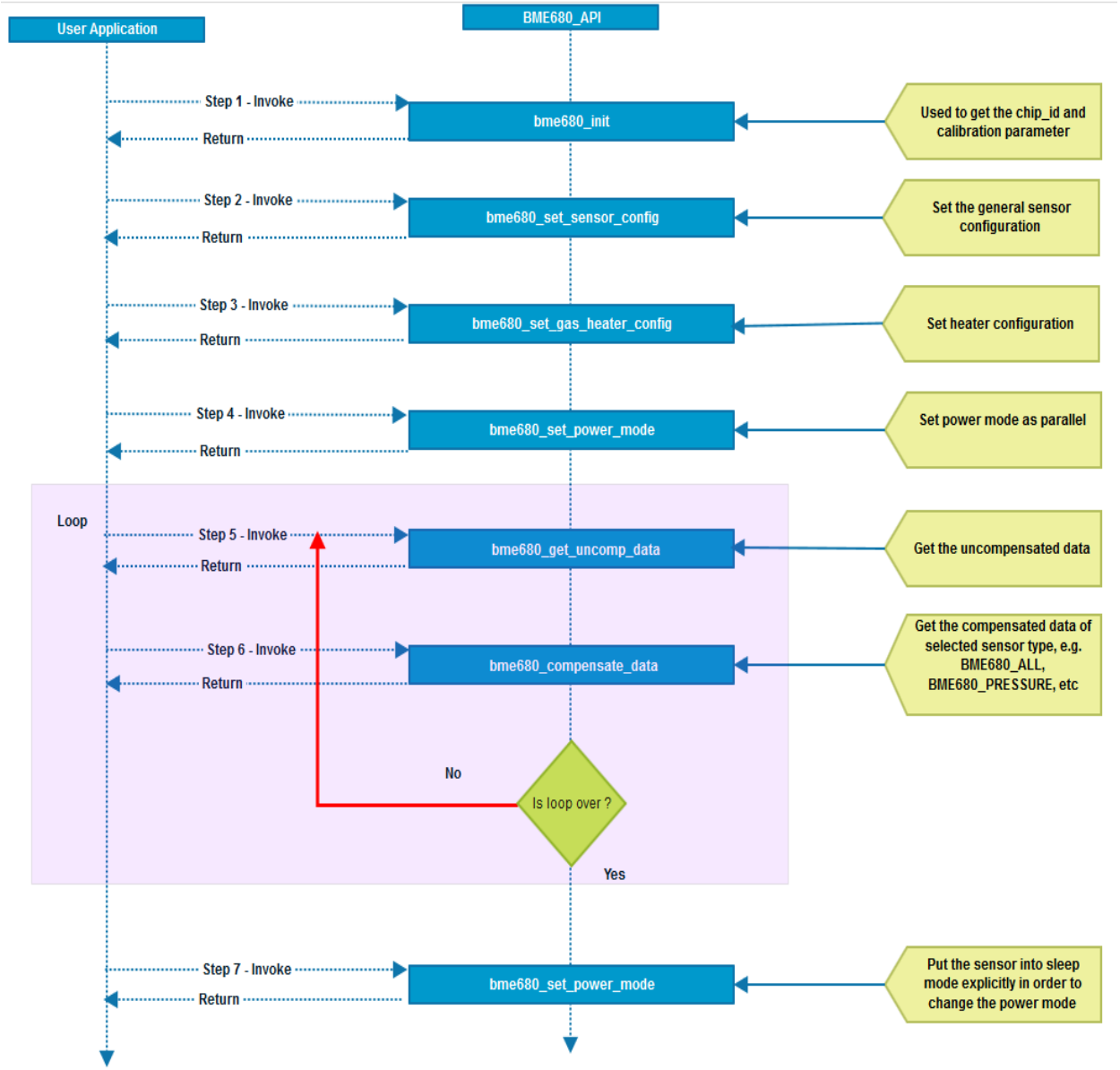


Figure 3 parallel mode

2.4.2 Example code

```
/* BME680 two sensor – parallel Mode – T+P+H+G Enabled */

/* include bme680 main header */
#include "bme680.h"
/*!
*       BME680_MAX_NO_OF_SENSOR = 2; defined in bme680.h file
*       Note: user can change sensor number according to requirements.
*       Test setup: It has been assumed that “BME680 sensor_0” interfaced over SPI and
*       “BME680 sensor_1” over I2C with device address 0x77(SDO level = HIGH)
*/
#define READ_COUNT 10
/* BME680 sensor structure instance */
struct bme680_t bme680_sensor_no[BME680_MAX_NO_OF_SENSOR];

/* BME680 sensor’s compensated data structure instance */
struct bme680_comp_field_data compensate_data_sensor[BME680_MAX_NO_OF_SENSOR][3];

/* BME680 sensor’s uncompensated data structure instance */
struct bme680_uncomp_field_data uncompensated_data_of_sensor[BME680_MAX_NO_OF_SENSOR][3];

/* BME680 sensor’s configuration structure instance */
struct bme680_sens_conf set_conf_sensor[BME680_MAX_NO_OF_SENSOR];

/* BME680 sensor’s heater configuration structure instance */
struct bme680_heater_conf set_heatr_conf_sensor[BME680_MAX_NO_OF_SENSOR];

void main(void)
{

        unsigned int i = BME680_INIT_VALUE;
        int j = BME680_INT_VALUE;
        enum bme680_return_type com_rslt = BME680_COMM_RES_ERROR;

        /* Do BME680 sensor structure instance initialization*/

        /* Sensor_0 interface over SPI with native chip select line */
        /* USER defined SPI bus read function       */
        bme680_sensor_no[0].bme680_bus_read = BME680_SPI_bus_read_user;
        /* USER defined SPI bus write function      */
        bme680_sensor_no[0].bme680_bus_write = BME680_SPI_bus_write_user;
        /* USER defined SPI burst read function */
        bme680_sensor_no[0].bme680_burst_read = BME680_SPI_bus_read_user;
        /* USER defined delay function */
        bme680_sensor_no[0].delay_msec = BME680_delay_msec_user;
        /* Mention communication interface */
        bme680_sensor_no[0].interface = BME680_SPI_INTERFACE;


        /* Sensor_1 interface over I2C with address as 0x77(SDO level = HIGH) */
        /* USER defined I2C bus write function      */
        bme680_sensor_no[1].bme680_bus_write = BME680_I2C_bus_write_user;
        /* USER defined I2C bus read function       */
        bme680_sensor_no[1].bme680_bus_read = BME680_I2C_bus_read_user;
        /* USER defined I2C burst read function */
        bme680_sensor_no[1].bme680_burst_read = BME680_I2C_bus_read_user;
        /* USER defined delay function      */
        bme680_sensor_no[1].delay_msec = BME680_delay_msec_user;
        /*set the address according to sensor's SDO level and interfaced protocol */
        bme680_sensor_no[1].dev_addr = BME680_I2C_ADDR_SECONDARY;
        /* Mention communication interface */
        bme680_sensor_no[1].interface = BME680_I2C_INTERFACE;
```

```
/* get chip id and calibration parameter */
com_rslt = bme680_init(&bme680_sensor_no[0]);
com_rslt += bme680_init(&bme680_sensor_no[1]);

/* Do Sensor initialization */
for (i=0; I < BME680_MAX_NO_OF_SENSOR; i++) {


        /* Check Device-ID before next steps of sensor operations */
        if (BME680_CHIP_ID == bme680_sensor_no[i].chip_id) {

        /* Select sensor configuration parameters */
        set_conf_sensor[i].heatr_ctrl = BME680_HEATR_CTRL_ENABLE;
        set_conf_sensor[i].odr = BME680_ODR_20MS;
        set_conf_sensor[i].run_gas = BME680_RUN_GAS_DISABLE;
        set_conf_sensor[i].nb_conv = 0x0A;
        set_conf_sensor[i].osrs_hum = BME680_OSRS_1X;
        set_conf_sensor[i].osrs_pres = BME680_OSRS_2X;
        set_conf_sensor[i].osrs_temp = BME680_OSRS_4X;
        set_conf_sensor[i].filter = BME680_FILTER_COEFF_1;


        /* activate sensor configuration */
        com_rslt += bme680_set_sensor_config(&set_conf_sensor[i],
                                             &bme680_sensor_no[i]);


        /* Select Heater configuration parameters */
        set_heatr_conf_sensor[i].heatr_dur[0] = 0;
        set_heatr_conf_sensor[i].heatr_dur[1] = 65536;
        set_heatr_conf_sensor[i].heatr_dur[2] = 89;
        set_heatr_conf_sensor[i].heatr_dur[3] = 90;
        set_heatr_conf_sensor[i].heatr_dur[4] = 91;
        set_heatr_conf_sensor[i].heatr_dur[5] = 92;
        set_heatr_conf_sensor[i].heatr_dur[6] = 93;
        set_heatr_conf_sensor[i].heatr_dur[7] = 94;
        set_heatr_conf_sensor[i].heatr_dur[8] = 95;
        set_heatr_conf_sensor[i].heatr_dur[9] = 96;

        set_heatr_conf_sensor[i].heatr_idacv[0] = 1;
        set_heatr_conf_sensor[i].heatr_idacv[1] = 2;
        set_heatr_conf_sensor[i].heatr_idacv[2] = 3;
        set_heatr_conf_sensor[i].heatr_idacv[3] = 4;
        set_heatr_conf_sensor[i].heatr_idacv[4] = 5;
        set_heatr_conf_sensor[i].heatr_idacv[5] = 6;
        set_heatr_conf_sensor[i].heatr_idacv[6] = 7;
        set_heatr_conf_sensor[i].heatr_idacv[7] = 8;
        set_heatr_conf_sensor[i].heatr_idacv[8] = 9;
        set_heatr_conf_sensor[i].heatr_idacv[9] = 10;

        set_heatr_conf_sensor[i].heater_temp[0] = 199;
        set_heatr_conf_sensor[i].heater_temp[1] = 200;
        set_heatr_conf_sensor[i].heater_temp[2] = 201;
        set_heatr_conf_sensor[i].heater_temp[3] = 247;
        set_heatr_conf_sensor[i].heater_temp[4] = 248;
        set_heatr_conf_sensor[i].heater_temp[5] = 249;
        set_heatr_conf_sensor[i].heater_temp[6] = 250;
        set_heatr_conf_sensor[i].heater_temp[7] = 399;
        set_heatr_conf_sensor[i].heater_temp[8] = 401;
        set_heatr_conf_sensor[i].heater_temp[9] = 400;

        set_heatr_conf_sensor[i].heatr_dur_shared = 200;
```

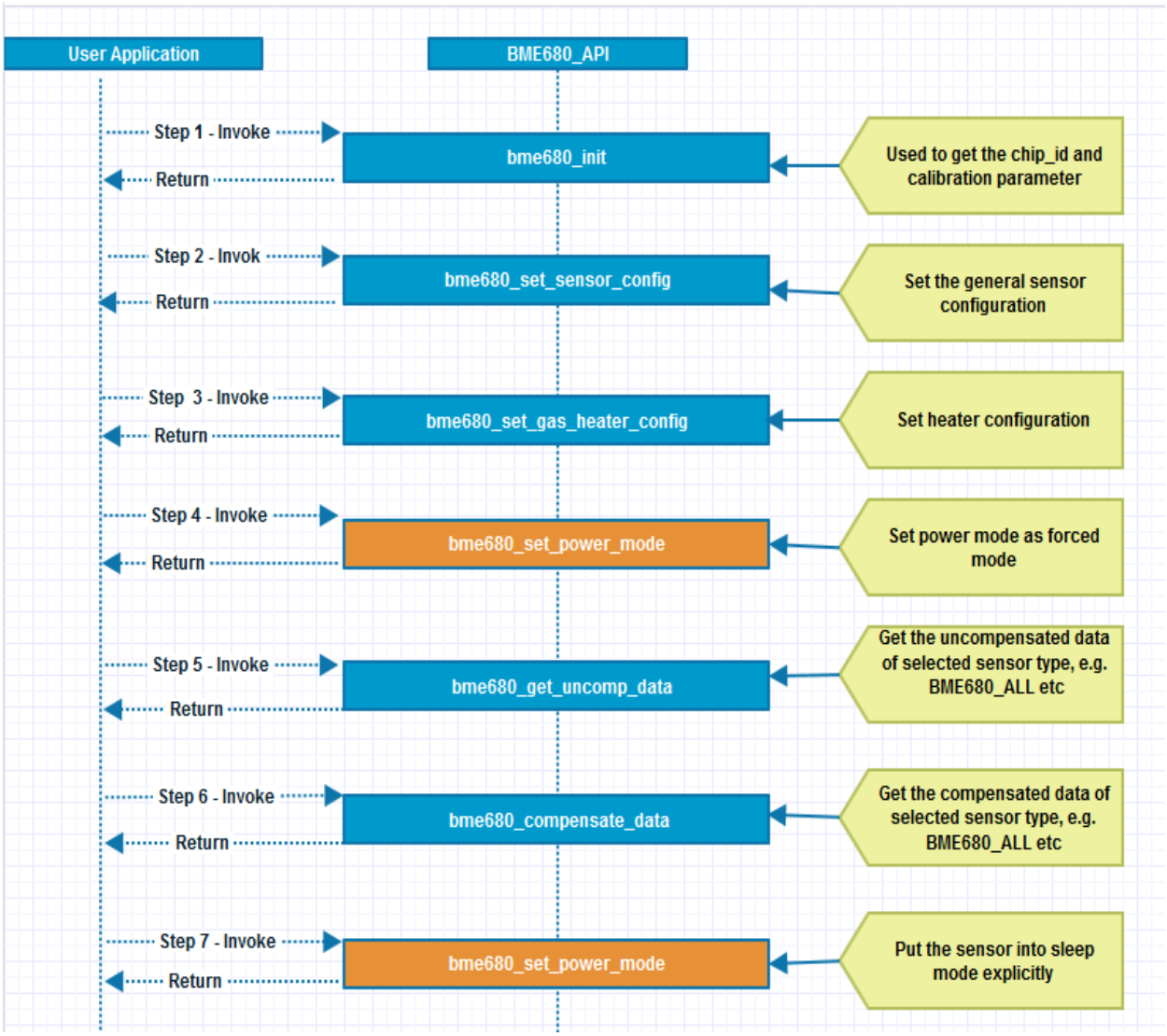
```
        set_heatr_conf_sensor[i].profile_cnt = 10;

        /* activate heater configuration */
        com_rslt += bme680_set_gas_heater_config(&set_heatr_conf_sensor[i],
        &bme680_sensor_no[i]);

        /* Set power mode as parallel mode */
        com_rslt += bme680_set_power_mode(BME680_PARALLEL_MODE,&bme680_sensor_no[i]);

if (BME680_COMM_RES_OK == com_rslt) {
        for (j = 0; j < READ_COUNT; j++) {

        /*Get the uncompensated T+P+G+H data*/
        bme680_get_uncomp_data(uncompensated_data_of_sensor[i], 3, BME680_ALL, &bme680_sen-
        sor_no[i]);

        /*Get the compensated T+P+G+H data*/
        bme680_compensate_data(uncompensated_data_of_sensor[i], compensate_data_sensor[i], 3,
        BME680_ALL, &bme680_sensor_no[i]);
        }
        /* put sensor into sleep mode explicitly */
        bme680_set_power_mode(BME680_SLEEP_MODE, &bme680_sensor_no[i]);

     }
   }
 }
}
```

NOTE : An user defined delay should be added in between setting the power mode (bme680_set_power_mode) and reading the data (bme680_get_uncomp_data) ,which is the sum of delays due to conversion of P, T, rH, Heater duration and gas resistance required for the proper data readout

2.5 How to change the sensor setting during Run-time

This example represent the use case of change in sensor setting during Run-time. It reads the compensated data of BME680 (two bme680 interfaced over I2C & SPI) Temperature, Pressure, Humidity & Gas sensor in forced mode operation with old and new sensor settings.

2.5.1 Sequence diagram



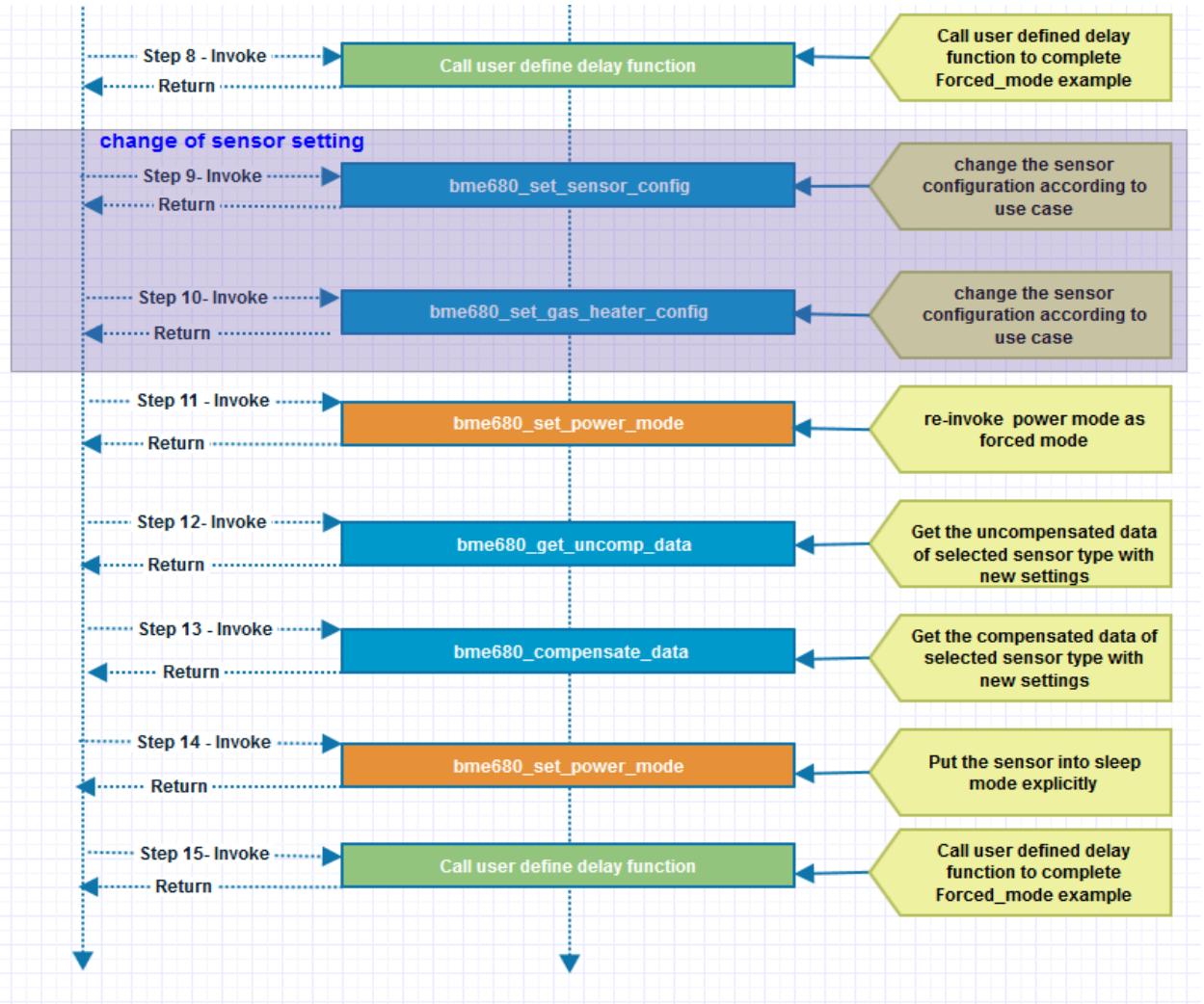


Figure 4 change of setting during run-time

2.5.1 Example code

```
/* BME680 sensor – Force Mode – T+P+H+G Enabled
   And change the sensor setting during run-time*/

/* include bme680 main header */
#include "bme680.h"
/*!
*       BME680_MAX_NO_OF_SENSOR = 2; defined in bme680.h file
*       Note: user can change sensor number according to requirements.
*       Test setup: It has been assumed that “BME680 sensor_0” interfaced over SPI and
*       “BME680 sensor_1” over I2C with device address 0x77(SDO level = HIGH)
*/



/* BME680 sensor structure instance */
struct bme680_t bme680_sensor_no[BME680_MAX_NO_OF_SENSOR];

/* BME680 sensor’s compensated data structure instance */
struct bme680_comp_field_data compensate_data_sensor[BME680_MAX_NO_OF_SENSOR][3];

/* BME680 sensor’s uncompensated data structure instance */
```

```
struct bme680_uncomp_field_data uncompensated_data_of_sensor[BME680_MAX_NO_OF_SENSOR][3];

/* BME680 sensor's configuration structure instance */
struct bme680_sens_conf set_conf_sensor[BME680_MAX_NO_OF_SENSOR];

/* BME680 sensor's heater configuration structure instance */
struct bme680_heater_conf set_heatr_conf_sensor[BME680_MAX_NO_OF_SENSOR];

void main(void)
{
	unsigned int i = BME680_INIT_VALUE;

	enum bme680_return_type com_rslt = BME680_COMM_RES_ERROR;

	/* Do BME680 sensor structure instance initialization*/

	/* Sensor_0 interface over SPI with native chip select line */
	/* USER defined SPI bus read function	*/
	bme680_sensor_no[0].bme680_bus_read = BME680_SPI_bus_read_user;
	/* USER defined SPI bus write function	*/
	bme680_sensor_no[0].bme680_bus_write = BME680_SPI_bus_write_user;
	/* USER defined SPI burst read function */
	bme680_sensor_no[0].bme680_burst_read = BME680_SPI_bus_read_user;
	/* USER defined delay function */
	bme680_sensor_no[0].delay_msec = BME680_delay_msec_user;
	/* Mention communication interface */
	bme680_sensor_no[0].interface = BME680_SPI_INTERFACE;


	/* Sensor_1 interface over I2C with address as 0x77(SDO level = HIGH) */
	/* USER defined I2C bus write function	*/
	bme680_sensor_no[1].bme680_bus_write = BME680_I2C_bus_write_user;
	/* USER defined I2C bus read function	*/
	bme680_sensor_no[1].bme680_bus_read = BME680_I2C_bus_read_user;
	/* USER defined I2C burst read function */
	bme680_sensor_no[1].bme680_burst_read = BME680_I2C_bus_read_user;
	/* USER defined delay function	*/
	bme680_sensor_no[1].delay_msec = BME680_delay_msec_user;
	/*set the address according to sensor's SDO level and interfaced protocol */
	bme680_sensor_no[1].dev_addr = BME680_I2C_ADDR_SECONDARY;
	/* Mention communication interface */
	bme680_sensor_no[1].interface = BME680_I2C_INTERFACE;

	/* get chip id and calibration parameter */
	com_rslt = bme680_init(&bme680_sensor_no[0]);
	com_rslt += bme680_init(&bme680_sensor_no[1]);

	/* Do Sensor initialization */
	for (i=0;i<BME680_MAX_NO_OF_SENSOR;i++) {


	/* Check Device-ID before next steps of sensor operations */
	if (BME680_CHIP_ID == bme680_sensor_no[i].chip_id) {

	/* Select sensor configuration parameters */
		set_conf_sensor[i].heatr_ctrl = BME680_HEATR_CTRL_ENABLE;
		set_conf_sensor[i].run_gas = BME680_RUN_GAS_ENABLE;
		set_conf_sensor[i].nb_conv = 0x00;
		set_conf_sensor[i].osrs_hum = BME680_OSRS_1X;
		set_conf_sensor[i].osrs_pres = BME680_OSRS_1X;
		set_conf_sensor[i].osrs_temp = BME680_OSRS_1X;

		/* activate sensor configuration */
```

```
        com_rslt += bme680_set_sensor_config(&set_conf_sensor[i],&bme680_sensor_no[i]);

        /* Select Heater configuration parameters */
        set_heatr_conf_sensor[i].heater_temp[0] = 300;
        set_heatr_conf_sensor[i].heatr_idacv[0] = 1;
        set_heatr_conf_sensor[i].heatr_dur[0] = 137;
        set_heatr_conf_sensor[i].profile_cnt = 1;

        /* activate heater configuration */
        com_rslt += bme680_set_gas_heater_config(&set_heatr_conf_sensor[i],
                                                 &bme680_sensor_no[i]);

        /* Set power mode as forced mode */
        com_rslt += bme680_set_power_mode(BME680_FORCED_MODE,&bme680_sensor_no[i]);

    if (BME680_COMM_RES_OK == com_rslt) {
        /*Get the uncompensated T+P+G+H data*/
        bme680_get_uncomp_data(uncompensated_data_of_sensor[i], 3, BME680_ALL,
        &bme680_sensor_no[i]);

        /*Get the compensated T+P+G+H data*/
        bme680_compensate_data(uncompensated_data_of_sensor[i], compensate_data_sen-
        sor[i], 3, BME680_ALL, &bme680_sensor_no[i]);

        /* put sensor into sleep mode explicitly */
        bme680_set_power_mode(BME680_SLEEP_MODE, &bme680_sensor_no[i]);

        /* call user define delay function(duration millisecond) */
        User_define_delay(100);

        /* change the oversampling setting of T+P+H */
        set_conf_sensor[i].osrs_hum = BME680_OSRS_2X;
        set_conf_sensor[i].osrs_pres = BME680_OSRS_2X;
        set_conf_sensor[i].osrs_temp = BME680_OSRS_2X;
        /* activate sensor configuration */
        com_rslt += bme680_set_sensor_config(&set_conf_sensor[i], &bme680_sen-
        sor_no[i]);

        /* change the heater duration setting */
        set_heatr_conf_sensor[i].heater_temp[0] = 250;
        set_heatr_conf_sensor[i].heatr_dur[0] = 137;

        /* activate heater configuration */
        com_rslt += bme680_set_gas_heater_config(&set_heatr_conf_sensor[i],
        &bme680_sensor_no[i]);

        /* Re-invoke power mode as forced mode */
        com_rslt += bme680_set_power_mode(BME680_FORCED_MODE, &bme680_sensor_no[i]);

        if (BME680_COMM_RES_OK == com_rslt) {
        /*Get the uncompensated T+P+G+H data*/
        bme680_get_uncomp_data(uncompensated_data_of_sensor[i], 3, BME680_ALL,
        &bme680_sensor_no[i]);

        /*Get the compensated T+P+G+H data*/

        bme680_compensate_data(uncompensated_data_of_sensor[i],
        compensate_data_sensor [i], 3, BME680_ALL, &bme680_sensor_no[i]);

        /* put sensor into sleep mode explicitly */
        bme680_set_power_mode(BME680_SLEEP_MODE, &bme680_sensor_no[i]);

        /* call user define delay function (duration millisecond) */
```

```
                User_define_delay(100);

            }
        }
    }
}
}
```

NOTE : An user defined delay should be added in between setting the power mode (bme680_set_power_mode) and reading the data (bme680_get_uncomp_data) ,which is the sum of delays due to conversion of P, T, rH, Heater duration and gas resistance required for the proper data readout

3 Sensor configuration

The below table represent settings of sensor in forced, sequential and parallel mode.

3.1 Sensor setting table

Forced mode	Sequential mode	Parallel mode
Select Filter for Pressure & Humidity • filter<2:0>	Set Wake period • odr <3>	Select Filter for Pressure & Humidity • filter<2:0>
Select Oversampling rating of Sensor • Set osrs_x • (P=16,T=2,rH=1)	Select Filter for Pressure & Humidity • filter<2:0>	Select Oversampling rating of Sensor • Set osrs_x • (P=16,T=2,rH=1)
Select temp source for temp1 & temp2 • Set temp2_en & ext_diode_sel • (Use TD1+ Internal diode)	Select Oversampling rating of Sensor • Set osrs_x • (P=16,T=2,rH=1)	Select temp source for temp1 & temp2 • Set temp2_en & ext_diode_sel • (Use TD1+ Internal diode)
Enable gas conversion • Set run_gas_l	Select temp source for temp1 & temp2 • Set temp2_en & ext_diode_sel • (Use TD1+ Internal diode)	Enable gas conversion • Set run_gas_l
Select index of profile that to be used • Set nb_conv (Numbering starts with 0)	Enable gas conversion • Set run_gas_l	Select number of profiles • Set nb_conv((Numbering starts with 1)
Set gas_wait for each profile • gas_wait_x <7:0>	Select number of profiles • Set nb_conv(Numbering starts with 1)	Set gas wait time between two T1T2PHG sub-measurement sequences • gas_wait_shared<7:0>
set res_heat for each profile • res_heat_x<7:0>	Set gas_wait for each profile • gas_wait_x <7:0>	set res_heat for each profile • res_heat_x<7:0>
Set Heater Current for each profile • idac_heat_x<7:0>	set res_heat for each profile • res_heat_x<7:0>	Set Heater Current for each profile • idac_heat_x<7:0>
Set mode to Forced mode	Set Heater Current for each profile	Set mode to Parallel mode

• mode<1:0>='01'	• idac_heat_x<7:0>	• mode<1:0>='10'
	Set mode to Sequential mode • mode<1:0>='11'	

3.2 SPI & I2C sample code

The sample code below represents the configuration of single BME680 sensor interfaced over SPI or I2C.

I2C BUS READ:

```
/*!     \Brief: The function is used as I2C bus read
*       \Return : Status of the I2C read
*       \param dev_addr : The device address of the sensor
*       \param reg_addr : Address of the first register, where data is to be read
*       \param reg_data : The data read from the sensor, which is held in an array
*       \param cnt : The no of data bytes to be read
*/

S8 BME680_I2C_bus_read(U8 dev_addr, U8 reg_addr, U8 *reg_data, U8 cnt)
{
        S32 iError=0;
        U8 array[I2C_BUFFER_LEN];
        U8 stringpos;
        array[0] = reg_addr;

        /*! Please add your I2C read function here with valid return value
        * iError is an return value of I2C read function
        * In the driver SUCCESS defined as 0 and FAILURE defined as -1 */

        for (stringpos = 0; stringpos < cnt; stringpos++)
        {
                *(reg_data + stringpos) = array[stringpos];
        }

        return (S8)iError;
}
```

I2C BUS WRITE:

```
/*!     \Brief: The function is used as I2C bus write
*       \Return : Status of the I2C write
*       \param dev_addr : The device address of the sensor
*       \param reg_addr : Address of the first register, where data is to be written
*       \param reg_data : The data to be written to the sensor's reg_addr, which is held in an
*              array
*       \param cnt : The no of byte of data to be written
*/
S8 BME680_I2C_bus_write(u8 dev_addr, u8 reg_addr, u8 *reg_data, u8 cnt)
{
        S32 iError=0;
        U8 array[I2C_BUFFER_LEN];
        U8 stringpos;
        array[0] = reg_addr;


        for (stringpos=0;stringpos<cnt;stringpos++)
        {
                array[stringpos+1] = *(reg_data + stringpos);
        }

        /* Please add your I2C write function here
        * iError is an return value of I2C write function
        * Please use valid return value
        * In the driver SUCCESS defined as 0 and FAILURE defined as -1 */

        return (S8)iError;
}
```

SPI BUS READ:

```
/*!     \Brief: The function is used for SPI bus read
*       \Return : Status of the SPI read
*       \param dev_addr : The device address of the sensor
*       \param reg_addr : Address of the first register, where data is going to be read
*       \param reg_data : The data read from the sensor, which is held in an array
*       \param cnt : The no of byte of data to be read
*/
S8 BME680_SPI_bus_read(U8 dev_addr, U8 reg_addr, U8 *reg_data, U8 cnt)
{

        S32 iError=0;
        U8 array[SPI_BUFFER_LEN]={0xFF};
```

```
	U8 stringpos;
		/*SPI mode has only 7 bits denoting the register address.
		The MSB of register address is the 8th bit which
		denotes the type of SPI operation read/write
		(read -> 1 / write -> 0)*/
	array[0] = reg_addr|0x80;
		/*read routine is initiated where the MSB is set to 1*/

	/* Please add your SPI read function here
	* iError is an return value of SPI read function
	* Please use valid return value
	* In the driver SUCCESS defined as 0 and FAILURE defined as -1 */

	for (stringpos = 0; stringpos < cnt;stringpos++)
	{
		*(reg_data + stringpos) = array[stringpos+1];
	}
	return (S8)iError;

}
```

SPI BUS WRITE:

```
/*!	\Brief: The function is used as SPI bus write
*	\Return : Status of the SPI write
*	\param dev_addr : The device address of the sensor
*	\param reg_addr : Address of the first register, will data is going to be written
*	\param reg_data : The data to be written in the sensor's reg_addr, which is held in an
*		array
*	\param cnt : The no of byte of data to be written
*/
S8 BME680_SPI_bus_write(U8 dev_addr, U8 reg_addr, U8 *reg_data, U8 cnt)
{
	S32 iError=0;
	U8 array[SPI_BUFFER_LEN*2];
	U8 stringpos;
		/*SPI mode has only 7 bits denoting the register address.
		The MSB of register address is the 8th bit which
		denotes the type of SPI operation read/write
		(read -> 1 / write -> 0)*/
	array[0] = (reg_addr & 0x7F);
		/*write routine is initiated where the MSB is set to 0 */

	for (stringpos = 0; stringpos < cnt; stringpos++)
	{
```

```
            array[stringpos+1] = *(reg_data + stringpos);
        }

        /* Please add your SPI write function here
        * iError is an return value of SPI write function
        * Please use valid return value
        * In the driver SUCCESS is defined as 0 and FAILURE is defined as -1 */

        return (S8)iError;
}
```

4 Frequently Asked Questions

Q1 What is BME680 sensor?

A. The BME680 is an ultra-low-power, low-voltage sensor used for high-precision pressure, temperature, humidity and gas measurements.

Q2 What are the communication interface supported?

A. Two interface protocols are available, SPI or I2C. Default interface is I2C.

Q3 How to select SPI interface?

A. The SPI interface is activated when CSB pad signal goes to low and deactivated when CSB pad signal goes back to high. The IO pads are switched to SPI mode at the falling edge of CSB signal and kept in SPI mode until a new power OFF/ON sequence occurs. The I2C interface is disabled by the falling edge of CSB pad signal and kept disabled until a new power OFF/ON sequence occurs.

Q4 What are the modes supported by SPI?

A. It supports mode 0 & 3.

Q5 Is 3-wire SPI interface supported?

A. Yes, The SPI interface has two modes: 4-wire and 3-wire. The protocol is the same for both to write. The 3- wire is selected by setting „1" to the register spi_3w_en. The pad SDI is used as a data pad in 3-wire mode.
Interface signals are following.

CSB chip select,	active low
SCK	clock
SDI	data input; data input/output in 3-wire mode
SDO	data output; Hi-Z level in 3-wire mode

Q6 What is the I2C device address?

A. Primary I2C add is 0x76(SDO = LOW) and secondary add = 0x77(SDO = HIGH)

Q7 Can I connect multiple sensor over selected interface?

A. Yes, this latest API supports multiple sensor integration. User can interface multiple sensor over any protocol or combination of I2C & SPI.

Q8 What is forced mode and how can I configure it?

A. In forced mode, the temperature, pressure, humidity, and gas conversions are performed sequentially for single measurement. When the measurement is finished, the sensor returns to sleep mode and the measurement results can be obtained from the data registers. For a next measurement, forced mode needs to be selected again.
Click here for forced mode example.

Q9 What is sequential mode and how can I configure it?

A. In sequential mode, the device periodically enters stand-by state and returns to an operational state after a given wake-up period. In the stand-by state, all analog blocks are powered down; the internal low frequency oscillator is enabled. After waking-up into the operational state, T, P, H and G measurements are performed sequentially in the following order (T1T2PHG0, T1T2PHG1, ..., T1T2PHG9), and the data registers are updated in the same order.
Click here for sequential mode example

Q10 What is parallel mode and how can I configure it?

A. In parallel mode, the device periodically performs sequences measuring T, P, H, and G physical quantities without entering the stand-by state. In parallel mode at least one of osrs_t|p|h is expected to be non-zero.
Click here for parallel mode example.

Q11 What is sleep mode and what is the significance of it?

A. Sleep mode is set by default after power on reset. In sleep mode, no measurements are performed and power consumption is at a minimum. All registers are accessible; Chip-ID and calibration parameters can be read.
User need to configure the sensor into sleep mode whenever change of power mode or change of sensor setting is requested. In forced mode operation, user must call sleep mode explicitly after completion of measurement.

Q12 How to enable and disable the T, P, H?

A. Enable & disable function of the sensor T, P and H is controlled by oversampling register osrs_x (2:0, x = t, p, h) respectively.

osrs_x(2:0)	Measurement
000	No measurement /skipped or disable
001	Oversampling x1
010	Oversampling x2
011	Oversampling x4
100	Oversampling x8
101, 110, 111	Oversampling x16

For detail check any example code.

Q13 Can I change the sensor configuration during RUN-TIME?

A. Change of sensor setting during active session is not desirable operation. User first put the sensor into the sleep mode then reconfigure the settings. After this, set the desirable power mode and perform the read operation.
Note: If user change the sensor setting during active session of measurements then it may produce the unexpected result.
Click here for example code.

Q14 Does the coding style of this API matches with "Linux"?

A. Yes, this API follows the coding standard guideline suggested by "**checkpatch.pl**" script file.

Q15 Does this API supports FIXED point compensation equation?

A. This API supports both fixed & floating point compensation. By default it supports floating point compensation. In order to use fixed point compensation user need to enable the macro **"FIXED_POINT_COMPENSATION"** in the bme680.h file.

Q16 Can I configure one sensor as **fixed point** and another sensor for **floating point** compensation over selected communication interface?

A. The current version of API only supports either FIXED or FLOATING point compensation for all the interfaced sensor. User don't have facility to configure one sensor as fixed point and another as floating point.

Q17 What is the use of "**BME680_SPECIFIC_FIELD_DATA_READ_ENABLED**" macro?

A. This macro is used for the compilation of sensor-type specific code.
If it is not defined in "bme680.h" then user will get uncompensated & compensated data for T, P, H, and G, thus any sensor type other than "BME680_ALL" will internally convert to BME680_ALL and prevent the compilation of sensor specific codes.
In order to get uncompensated & compensated data of a specific sensor type (**BME680_TEMPERATURE, BME680_PRESSURE, BME680_HUMIDITY, and BME680_GAS**) user need to define the "BME680_SPECIFIC_FIELD_DATA_READ_ENABLED" macro in bme680.h file.

5 Miscellaneous

5.1 Assumptions

It has been assumed that user is having the basic understanding of BME680 and communication protocol interface.

5.2 Constraints

Selection of interface (SPI & I2C) driver according to user application and its initialization.

5.3 Dependencies

Accuracy of data is subject to sensor's quality.

5.4 Out of scope

1. Communication driver related dependencies.

5.5 Key Performance Indicators

The memory footprint measurement shall be performed with the below compiler and configuration

a. Compiler : GCC
b. IDE : Eclipse
c. Options : O2 (Optimization level)
d. ROM size (text/code + data)

Configuration	***Size in KB***
All configurations enabled	***7.08***
Disable BME680_SPECIFIC_FIELD_DATA_READ_ENABLED	***5.4***

e. RAM size (data + bss) : NIL