## Table of Contents

- [Introduction](#intro)

- [Version](#ver)

- [Integration details](#integration)

- [Driver files information](#fileinfo)

- [Supported sensor interface](#interface)

- [Copyright](#copy)

### Introduction<a name=intro></a>

- This package contains the Bosch Sensortec MEMS accelerometer sensor driver (sensor API)

- The sensor driver package includes bma2x2.h, bma2x2.c and bma2x2\_support.c files

- BMA2x2 sensor driver supports the following Bosch MEMS sensors

\* BMA280

\* BMA255

\* BMA253

\* BMA250E

\* BMA22E

\* BMA220

\* BMI055 - Combination of bma2x2 + bmg160 APIs

\* BMX055 - Combination of bma2x2 + bmg160 + bmm050 APIs

\* BMC150 - Combination of bma2x2 + bmm050 APIs

\* BMC056 - Combination of bma2x2 + bmm050 APIs

### Version<a name=ver></a>

- Version of bma2x2 sensor driver is:

Driver files | Version

-----------------|---------

bma2x2.c | 2.0.7

bma2x2.h | 2.0.7

bma2x2\_support.c | 1.0.4

### Integration details<a name=integration></a>

- Integrate bma2x2.h and bma2x2.c file in to your project.

- The bma2x2\_support.c file contains only examples for API use cases, so it is not required to integrate into project.

### Driver files information<a name=fileinfo></a>

- bma2x2.h

- This header file has the register address definition, constant definitions, data type definition and supported sensor driver calls declarations.

- bma2x2.c

- This file contains the implementation for the sensor driver APIs.

- bma2x2\_support.c

- This file shall be used as an user guidance, here you can find samples of

- Initialize the sensor with I2C/SPI communication

- Add your code to the SPI and/or I2C bus read and bus write functions.

- Return value can be chosen by yourself

- API just passes that value to your application code

- Add your code to the delay function

- Change I2C address accordingly in bma2x2.h

- Power mode configuration of the sensor

- Get and set functions usage

- Reading the sensor read out data

### Supported sensor interface<a name=interface></a>

- This accelerometer sensor driver supports SPI and I2C interfaces

### Copyright<a name=copy></a>

- Copyright (C) 2015 - 2016 Bosch Sensortec GmbH

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Copyright (C) 2015 - 2016 Bosch Sensortec GmbH

\*

\* bma2x2\_support.c

\* Date: 2016/03/09

\* Revision: 1.0.4 $

\*

\* Usage: Sensor Driver support file for BMA2x2 sensor

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Disclaimer

\*

\* Common:

\* Bosch Sensortec products are developed for the consumer goods industry.

\* They may only be used within the parameters of the respective valid

\* product data sheet. Bosch Sensortec products are provided with the

\* express understanding that there is no warranty of fitness for a

\* particular purpose.They are not fit for use in life-sustaining,

\* safety or security sensitive systems or any system or device

\* that may lead to bodily harm or property damage if the system

\* or device malfunctions. In addition,Bosch Sensortec products are

\* not fit for use in products which interact with motor vehicle systems.

\* The resale and or use of products are at the purchasers own risk and

\* his own responsibility. The examination of fitness for the intended use

\* is the sole responsibility of the Purchaser.

\*

\* The purchaser shall indemnify Bosch Sensortec from all third party

\* claims, including any claims for incidental, or consequential damages,

\* arising from any product use not covered by the parameters of

\* the respective valid product data sheet or not approved by

\* Bosch Sensortec and reimburse Bosch Sensortec for all costs in

\* connection with such claims.

\*

\* The purchaser must monitor the market for the purchased products,

\* particularly with regard to product safety and inform Bosch Sensortec

\* without delay of all security relevant incidents.

\*

\* Engineering Samples are marked with an asterisk (\*) or (e).

\* Samples may vary from the valid technical specifications of the product

\* series. They are therefore not intended or fit for resale to third

\* parties or for use in end products. Their sole purpose is internal

\* client testing. The testing of an engineering sample may in no way

\* replace the testing of a product series. Bosch Sensortec assumes

\* no liability for the use of engineering samples.

\* By accepting the engineering samples, the Purchaser agrees to indemnify

\* Bosch Sensortec from all claims arising from the use of engineering

\* samples.

\*

\* Special:

\* This software module (hereinafter called "Software") and any information

\* on application-sheets (hereinafter called "Information") is provided

\* free of charge for the sole purpose to support your application work.

\* The Software and Information is subject to the following

\* terms and conditions:

\*

\* The Software is specifically designed for the exclusive use for

\* Bosch Sensortec products by personnel who have special experience

\* and training. Do not use this Software if you do not have the

\* proper experience or training.

\*

\* This Software package is provided `` as is `` and without any expressed

\* or implied warranties,including without limitation, the implied warranties

\* of merchantability and fitness for a particular purpose.

\*

\* Bosch Sensortec and their representatives and agents deny any liability

\* for the functional impairment

\* of this Software in terms of fitness, performance and safety.

\* Bosch Sensortec and their representatives and agents shall not be liable

\* for any direct or indirect damages or injury, except as

\* otherwise stipulated in mandatory applicable law.

\*

\* The Information provided is believed to be accurate and reliable.

\* Bosch Sensortec assumes no responsibility for the consequences of use

\* of such Information nor for any infringement of patents or

\* other rights of third parties which may result from its use.

\* No license is granted by implication or otherwise under any patent or

\* patent rights of Bosch. Specifications mentioned in the Information are

\* subject to change without notice.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*---------------------------------------------------------------------------\*/

/\* Includes\*/

/\*---------------------------------------------------------------------------\*/

#include "bma2x2.h"

/\*----------------------------------------------------------------------------\*

\* The following functions are used for reading and writing of

\* sensor data using I2C or SPI communication

\*----------------------------------------------------------------------------\*/

#ifdef BMA2x2\_API

/\* \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,

\* will data is going to be read

\* \param reg\_data : This data read from the sensor,

\* which is hold in an array

\* \param cnt : The no of byte of data to be read

\*/

s8 BMA2x2\_I2C\_bus\_read(u8 dev\_addr, u8 reg\_addr, u8 \*reg\_data, u8 cnt);

/\* \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,

\* will data is going to be written

\* \param reg\_data : It is a value hold in the array,

\* will be used for write the value into the register

\* \param cnt : The no of byte of data to be write

\*/

s8 BMA2x2\_I2C\_bus\_write(u8 dev\_addr, u8 reg\_addr, u8 \*reg\_data, u8 cnt);

/\* \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 : It is a value hold in the array,

\* will be used for write the value into the register

\* \param cnt : The no of byte of data to be write

\*/

s8 BMA2x2\_SPI\_bus\_write(u8 dev\_addr, u8 reg\_addr, u8 \*reg\_data, u8 cnt);

/\* \Brief: The function is used as 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,

\* will data is going to be read

\* \param reg\_data : This data read from the sensor, which is hold in an array

\* \param cnt : The no of byte of data to be read \*/

s8 BMA2x2\_SPI\_bus\_read(u8 dev\_addr, u8 reg\_addr, u8 \*reg\_data, u8 cnt);

/\*

\* \Brief: SPI/I2C init routine

\*/

s8 I2C\_routine(void);

s8 SPI\_routine(void);

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of I2C/SPI function declarations\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Brief : The delay routine

\* \param : delay in ms

\*/

void BMA2x2\_delay\_msek(u32 msek);

/\*!

\* @brief This function is an example for delay

\* @param : None

\* @return : communication result

\*/

s32 bma2x2\_data\_readout\_template(void);

/\*----------------------------------------------------------------------------\*

\* struct bma2x2\_t parameters can be accessed by using bma2x2

\* bma2x2\_t having the following parameters

\* Bus write function pointer: BMA2x2\_WR\_FUNC\_PTR

\* Bus read function pointer: BMA2x2\_RD\_FUNC\_PTR

\* Burst read function pointer: BMA2x2\_BRD\_FUNC\_PTR

\* Delay function pointer: delay\_msec

\* I2C address: dev\_addr

\* Chip id of the sensor: chip\_id

\*---------------------------------------------------------------------------\*/

struct bma2x2\_t bma2x2;

/\*----------------------------------------------------------------------------\*

\* V\_BMA2x2RESOLUTION\_u8R used for selecting the accelerometer resolution

\* 12 bit

\* 14 bit

\* 10 bit

\*----------------------------------------------------------------------------\*/

extern u8 V\_BMA2x2RESOLUTION\_u8R;

/\* This function is an example for reading sensor data

\* \param: None

\* \return: communication result

\*/

s32 bma2x2\_data\_readout\_template(void)

{

/\*Local variables for reading accel x, y and z data\*/

s16 accel\_x\_s16, accel\_y\_s16, accel\_z\_s16 = BMA2x2\_INIT\_VALUE;

/\* bma2x2acc\_data structure used to read accel xyz data\*/

struct bma2x2\_accel\_data sample\_xyz;

/\* bma2x2acc\_data\_temp structure used to read

accel xyz and temperature data\*/

struct bma2x2\_accel\_data\_temp sample\_xyzt;

/\* Local variable used to assign the bandwidth value\*/

u8 bw\_value\_u8 = BMA2x2\_INIT\_VALUE;

/\* Local variable used to set the bandwidth value\*/

u8 banwid = BMA2x2\_INIT\_VALUE;

/\* status of communication\*/

s32 com\_rslt = ERROR;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* START INITIALIZATION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Based on the user need configure I2C or SPI interface.

\* It is example code to explain how to use the bma2x2 API\*/

#ifdef BMA2x2\_API

I2C\_routine();

/\*SPI\_routine(); \*/

#endif

/\*--------------------------------------------------------------------------\*

\* This function used to assign the value/reference of

\* the following parameters

\* I2C address

\* Bus Write

\* Bus read

\* Chip id

\*-------------------------------------------------------------------------\*/

com\_rslt = bma2x2\_init(&bma2x2);

/\* For initialization it is required to set the mode of

\* the sensor as "NORMAL"

\* NORMAL mode is set from the register 0x11 and 0x12

\* 0x11 -> bit 5,6,7 -> set value as 0

\* 0x12 -> bit 5,6 -> set value as 0

\* data acquisition/read/write is possible in this mode

\* by using the below API able to set the power mode as NORMAL

\* For the Normal/standby/Low power 2 mode Idle time

of at least 2us(micro seconds)

\* required for read/write operations\*/

/\* Set the power mode as NORMAL\*/

com\_rslt += bma2x2\_set\_power\_mode(BMA2x2\_MODE\_NORMAL);

/\* Note:

\* For the Suspend/Low power1 mode Idle time of

at least 450us(micro seconds)

\* required for read/write operations\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END INITIALIZATION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*------------------------------------------------------------------------\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* START GET and SET FUNCTIONS DATA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*---------------------------------------------------------------------------\*/

/\* This API used to Write the bandwidth of the sensor input

value have to be given

bandwidth is set from the register 0x10 bits from 1 to 4\*/

bw\_value\_u8 = 0x08;/\* set bandwidth of 7.81Hz\*/

com\_rslt += bma2x2\_set\_bw(bw\_value\_u8);

/\* This API used to read back the written value of bandwidth\*/

com\_rslt += bma2x2\_get\_bw(&banwid);

/\*-----------------------------------------------------------------\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END GET and SET FUNCTIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*-------------------------------------------------------------------\*/

/\*------------------------------------------------------------------\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* START READ SENSOR DATA(X,Y and Z axis) \*\*\*\*\*\*\*\*

\*---------------------------------------------------------------------\*/

/\* Read the accel X data\*/

com\_rslt += bma2x2\_read\_accel\_x(&accel\_x\_s16);

/\* Read the accel Y data\*/

com\_rslt += bma2x2\_read\_accel\_y(&accel\_y\_s16);

/\* Read the accel Z data\*/

com\_rslt += bma2x2\_read\_accel\_z(&accel\_z\_s16);

/\* accessing the bma2x2acc\_data parameter by using sample\_xyz\*/

/\* Read the accel XYZ data\*/

com\_rslt += bma2x2\_read\_accel\_xyz(&sample\_xyz);

/\* accessing the bma2x2acc\_data\_temp parameter by using sample\_xyzt\*/

/\* Read the accel XYZT data\*/

com\_rslt += bma2x2\_read\_accel\_xyzt(&sample\_xyzt);

/\*--------------------------------------------------------------------\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END READ SENSOR DATA(X,Y and Z axis) \*\*\*\*\*\*\*\*\*\*\*\*

\*-------------------------------------------------------------------------\*/

/\*-----------------------------------------------------------------------\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* START DE-INITIALIZATION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*-------------------------------------------------------------------------\*/

/\* For de-initialization it is required to set the mode of

\* the sensor as "DEEP SUSPEND"

\* DEEP SUSPEND mode is set from the register 0x11

\* 0x11 -> bit 5 -> set value as 1

\* the device reaches the lowest power consumption only

\* interface selection is kept alive

\* No data acquisition is performed

\* by using the below API able to set the power mode as DEEPSUSPEND\*/

/\* Set the power mode as DEEPSUSPEND\*/

com\_rslt += bma2x2\_set\_power\_mode(BMA2x2\_MODE\_DEEP\_SUSPEND);

/\*---------------------------------------------------------------------\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END DE-INITIALIZATION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*---------------------------------------------------------------------\*/

return com\_rslt;

}

#ifdef BMA2x2\_API

/\*--------------------------------------------------------------------------\*

\* The following function is used to map the I2C bus read, write, delay and

\* device address with global structure bma2x2\_t

\*-------------------------------------------------------------------------\*/

s8 I2C\_routine(void)

{

/\*--------------------------------------------------------------------------\*

\* By using bma2x2 the following structure parameter can be accessed

\* Bus write function pointer: BMA2x2\_WR\_FUNC\_PTR

\* Bus read function pointer: BMA2x2\_RD\_FUNC\_PTR

\* Delay function pointer: delay\_msec

\* I2C address: dev\_addr

\*--------------------------------------------------------------------------\*/

bma2x2.bus\_write = BMA2x2\_I2C\_bus\_write;

bma2x2.bus\_read = BMA2x2\_I2C\_bus\_read;

bma2x2.delay\_msec = BMA2x2\_delay\_msek;

bma2x2.dev\_addr = BMA2x2\_I2C\_ADDR2;

return BMA2x2\_INIT\_VALUE;

}

/\*---------------------------------------------------------------------------\*

\* The following function is used to map the SPI bus read, write and delay

\* with global structure bma2x2\_t

\*--------------------------------------------------------------------------\*/

s8 SPI\_routine(void)

{

/\*--------------------------------------------------------------------------\*

\* By using bma2x2 the following structure parameter can be accessed

\* Bus write function pointer: BMA2x2\_WR\_FUNC\_PTR

\* Bus read function pointer: BMA2x2\_RD\_FUNC\_PTR

\* Delay function pointer: delay\_msec

\*--------------------------------------------------------------------------\*/

bma2x2.bus\_write = BMA2x2\_SPI\_bus\_write;

bma2x2.bus\_read = BMA2x2\_SPI\_bus\_read;

bma2x2.delay\_msec = BMA2x2\_delay\_msek;

return BMA2x2\_INIT\_VALUE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\* I2C/SPI buffer length \*\*\*\*\*\*/

#define I2C\_BUFFER\_LEN 8

#define SPI\_BUFFER\_LEN 5

#define BMA2x2\_BUS\_READ\_WRITE\_ARRAY\_INDEX 1

#define BMA2x2\_SPI\_BUS\_WRITE\_CONTROL\_BYTE 0x7F

#define BMA2x2\_SPI\_BUS\_READ\_CONTROL\_BYTE 0x80

/\*-------------------------------------------------------------------\*

\* This is a sample code for read and write the data by using I2C/SPI

\* Use either I2C or SPI based on your need

\*

\*-----------------------------------------------------------------------\*/

/\* For configuring the I2C it is required to switch ON

\* SDI, SDO and CLk and also select the device address

\* The following definition of I2C address is used for the following sensors

\* BMA255

\* BMA253

\* BMA355

\* BMA280

\* BMA282

\* BMA223

\* BMA254

\* BMA284

\* BMA250E

\* BMA222E

#define BMA2x2\_I2C\_ADDR1 0x18

#define BMA2x2\_I2C\_ADDR2 0x19

\* The following definition of I2C address is used for the following sensors

\* BMC150

\* BMC056

\* BMC156

#define BMA2x2\_I2C\_ADDR3 0x10

#define BMA2x2\_I2C\_ADDR4 0x11

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \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,

\* will data is going to be written

\* \param reg\_data : It is a value hold in the array,

\* will be used for write the value into the register

\* \param cnt : The no of byte of data to be write

\*/

s8 BMA2x2\_I2C\_bus\_write(u8 dev\_addr, u8 reg\_addr, u8 \*reg\_data, u8 cnt)

{

s32 iError = BMA2x2\_INIT\_VALUE;

u8 array[I2C\_BUFFER\_LEN];

u8 stringpos = BMA2x2\_INIT\_VALUE;

array[BMA2x2\_INIT\_VALUE] = reg\_addr;

for (stringpos = BMA2x2\_INIT\_VALUE; stringpos < cnt; stringpos++) {

array[stringpos + BMA2x2\_BUS\_READ\_WRITE\_ARRAY\_INDEX] =

\*(reg\_data + stringpos);

}

/\*

\* Please take the below function as your reference for

\* write the data using I2C communication

\* "IERROR = I2C\_WRITE\_STRING(DEV\_ADDR, ARRAY, CNT+1)"

\* add your I2C write function here

\* iError is an return value of I2C read function

\* Please select your valid return value

\* In the driver SUCCESS defined as 0

\* and FAILURE defined as -1

\* Note :

\* This is a full duplex operation,

\* The first read data is discarded, for that extra write operation

\* have to be initiated. For that cnt+1 operation

\* done in the I2C write string function

\* For more information please refer data sheet SPI communication:

\*/

return (s8)iError;

}

/\* \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,

\* will data is going to be read

\* \param reg\_data : This data read from the sensor,

\* which is hold in an array

\* \param cnt : The no of byte of data to be read

\*/

s8 BMA2x2\_I2C\_bus\_read(u8 dev\_addr, u8 reg\_addr, u8 \*reg\_data, u8 cnt)

{

s32 iError = BMA2x2\_INIT\_VALUE;

u8 array[I2C\_BUFFER\_LEN] = {BMA2x2\_INIT\_VALUE};

u8 stringpos = BMA2x2\_INIT\_VALUE;

array[BMA2x2\_INIT\_VALUE] = reg\_addr;

/\* Please take the below function as your reference

\* for read the data using I2C communication

\* add your I2C rad function here.

\* "IERROR = I2C\_WRITE\_READ\_STRING(DEV\_ADDR, ARRAY, ARRAY, 1, CNT)"

\* iError is an return value of SPI write function

\* Please select your valid return value

\* In the driver SUCCESS defined as 0

\* and FAILURE defined as -1

\*/

for (stringpos = BMA2x2\_INIT\_VALUE; stringpos < cnt; stringpos++)

\*(reg\_data + stringpos) = array[stringpos];

return (s8)iError;

}

/\* \Brief: The function is used as 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,

\* will data is going to be read

\* \param reg\_data : This data read from the sensor,

\* which is hold in an array

\* \param cnt : The no of byte of data to be read \*/

s8 BMA2x2\_SPI\_bus\_read(u8 dev\_addr, u8 reg\_addr, u8 \*reg\_data, u8 cnt)

{

s32 iError = BMA2x2\_INIT\_VALUE;

u8 array[SPI\_BUFFER\_LEN] = {0xFF};

u8 stringpos;

/\* For the SPI mode only 7 bits of register addresses are used.

The MSB of register address is declared the bit what functionality it is

read/write (read as 1/write as 0)\*/

array[BMA2x2\_INIT\_VALUE] = reg\_addr|BMA2x2\_SPI\_BUS\_READ\_CONTROL\_BYTE;

/\*read routine is initiated register address is mask with 0x80\*/

/\*

\* Please take the below function as your reference for

\* read the data using SPI communication

\* " IERROR = SPI\_READ\_WRITE\_STRING(ARRAY, ARRAY, CNT+1)"

\* add your SPI read function here

\* iError is an return value of SPI read function

\* Please select your valid return value

\* In the driver SUCCESS defined as 0

\* and FAILURE defined as -1

\* Note :

\* This is a full duplex operation,

\* The first read data is discarded, for that extra write operation

\* have to be initiated. For that cnt+1 operation done in the SPI read

\* and write string function

\* For more information please refer data sheet SPI communication:

\*/

for (stringpos = BMA2x2\_INIT\_VALUE; stringpos < cnt; stringpos++) {

\*(reg\_data + stringpos) = array[stringpos +

BMA2x2\_BUS\_READ\_WRITE\_ARRAY\_INDEX];

}

return (s8)iError;

}

/\* \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 : It is a value hold in the array,

\* will be used for write the value into the register

\* \param cnt : The no of byte of data to be write

\*/

s8 BMA2x2\_SPI\_bus\_write(u8 dev\_addr, u8 reg\_addr, u8 \*reg\_data, u8 cnt)

{

s32 iError = BMA2x2\_INIT\_VALUE;

u8 array[SPI\_BUFFER\_LEN \* 2];

u8 stringpos = BMA2x2\_INIT\_VALUE;

for (stringpos = BMA2x2\_INIT\_VALUE; stringpos < cnt; stringpos++) {

/\* the operation of (reg\_addr++)&0x7F done:

because it ensure the

0 and 1 of the given value

It is done only for 8bit operation\*/

array[stringpos \* 2] = (reg\_addr++) &

BMA2x2\_SPI\_BUS\_WRITE\_CONTROL\_BYTE;

array[stringpos \* 2 + BMA2x2\_BUS\_READ\_WRITE\_ARRAY\_INDEX] =

\*(reg\_data + stringpos);

}

/\* Please take the below function as your reference

\* for write the data using SPI communication

\* add your SPI write function here.

\* "IERROR = SPI\_WRITE\_STRING(ARRAY, CNT\*2)"

\* iError is an return value of SPI write function

\* Please select your valid return value

\* In the driver SUCCESS defined as 0

\* and FAILURE defined as -1

\*/

return (s8)iError;

}

/\* Brief : The delay routine

\* \param : delay in ms

\*/

void BMA2x2\_delay\_msek(u32 msek)

{

/\*Here you can write your own delay routine\*/

}

#endif