```c
/**
* Copyright (c) 2020 Bosch Sensortec GmbH. All rights reserved.
*
* BSD-3-Clause
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the distribution.
*
* 3. Neither the name of the copyright holder nor the names of its
*    contributors may be used to endorse or promote products derived from
*    this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
* FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
* COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
* SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
* IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
* POSSIBILITY OF SUCH DAMAGE.
*
* @file bno055_support.c
* @date 10/01/2020
* @version  2.0.6
*
*/

/*---------------------------------------------------------------------------*
*  Includes
*---------------------------------------------------------------------------*/
#include "app.h"
#include "bno055.h"
#include "bno055_support.h"
#include "Mc32_I2cUtilCCS.h"
#include "driver/tmr/drv_tmr_static.h"

// Global variable
TIMER_DATA timerData;

#ifdef  BNO055_API

s32 bno055_read_routine(s_bno055_data *data)
{
      /* Variable used to return value of
     * communication routine*/
    s32 comres = BNO055_ERROR;

    /* variable used to set the power mode of the sensor*/
    //u8 power_mode = BNO055_INIT_VALUE;

    /*  For initializing the BNO sensor it is required to the operation mode
     * of the sensor as NORMAL
     * Normal mode can set from the register
     * Page - page0
     * register - 0x3E
     * bit positions - 0 and 1*/
    //power_mode = BNO055_POWER_MODE_NORMAL;

    /* set the power mode as NORMAL*/
    //comres += bno055_set_power_mode(power_mode);

    /*-----------------------------------------------------------------*
     ************************* END INITIALIZATION *************************
     *-----------------------------------------------------------------*/

    /************************* START READ RAW FUSION DATA ********
     * For reading fusion data it is required to set the
     * operation modes of the sensor
     * operation mode can set from the register
     * page - page0
     * register - 0x3D
     * bit - 0 to 3
     * for sensor data read following operation mode have to set
     * FUSION MODE
     * 0x08 - BNO055_OPERATION_MODE_IMUPLUS
     * 0x09 - BNO055_OPERATION_MODE_COMPASS
     * 0x0A - BNO055_OPERATION_MODE_M4G
     * 0x0B - BNO055_OPERATION_MODE_NDOF_FMC_OFF
     * 0x0C - BNO055_OPERATION_MODE_NDOF
     * based on the user need configure the operation mode*/
    //comres += bno055_set_operation_mode(BNO055_OPERATION_MODE_NDOF);

    /*  Raw Quaternion W, X, Y and Z data can read from the register
     * page - page 0
     * register - 0x20 to 0x27 */
    comres += bno055_read_quaternion_wxyz(&data->quaternion);
    /************************* END READ RAW FUSION DATA  ************/
    /******************START READ CONVERTED SENSOR DATA****************/
     /*  API used to read mag data output as double  - uT(micro Tesla)
      * float functions also available in the BNO055 API */
     comres += bno055_convert_double_mag_xyz_uT(&data->mag);
     /*  API used to read gyro data output as double  - dps and rps
      * float functions also available in the BNO055 API */
     comres += bno055_convert_double_gyro_xyz_dps(&data->gyro);
     /*  API used to read Euler data output as double  - degree and radians
      * float functions also available in the BNO055 API */
     comres += bno055_convert_double_euler_hpr_deg(&data->euler);
     /*  API used to read Linear acceleration data output as m/s2
      * float functions also available in the BNO055 API */
     comres += bno055_convert_double_linear_accel_xyz_msq(&data->linear_accel);
     comres += bno055_convert_double_gravity_xyz_msq(&data->gravity);

     /*-----------------------------------------------------------------*
      ************************* START DE-INITIALIZATION ***********************
      *-----------------------------------------------------------------*/

     /*  For de - initializing the BNO sensor it is required
      * to the operation mode of the sensor as SUSPEND
      * Suspend mode can set from the register
      * Page - page0
      * register - 0x3E
      * bit positions - 0 and 1*/
     //power_mode = BNO055_POWER_MODE_SUSPEND;

     /* set the power mode as SUSPEND*/
     //comres += bno055_set_power_mode(power_mode);

     /* Flag measure ready */
     data->flagMeasReady = true;

     /*-----------------------------------------------------------------*
     ************************* END DE-INITIALIZATION **********************
     *-----------------------------------------------------------------*/
     return (comres+1);
}

/*--------------------------------------------------------------------------*
 *  The following API is used to map the I2C bus read, write, delay and
 *  device address with global structure bno055_t
 *-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*
 *  By using bno055 the following structure parameter can be accessed
 *  Bus write function pointer: BNO055_WR_FUNC_PTR
 *  Bus read function pointer: BNO055_RD_FUNC_PTR
 *  Delay function pointer: delay_msec
 *  I2C address: dev_addr
 *-------------------------------------------------------------------------*/
s8 I2C_routine(void)
{
    bno055.bus_write = BNO055_I2C_bus_write;
    bno055.bus_read = BNO055_I2C_bus_read;
    bno055.delay_msec = BNO055_delay_msek;
    bno055.dev_addr = BNO055_I2C_ADDR1;
    return BNO055_INIT_VALUE;
}

/************** I2C buffer length******/

#define I2C_BUFFER_LEN 8
#define I2C0           5

/*-------------------------------------------------------------------*
 *
 *  This is a sample code for read and write the data by using I2C
 *  Use either I2C  based on your need
 *  The device address defined in the bno055.h file
 *
 *--------------------------------------------------------------------*/

/*  \Brief: The API 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 BNO055_I2C_bus_write(u8 dev_addr, u8 reg_addr, u8 *reg_data, u8 cnt)
{
    s8 BNO055_iERROR = BNO055_INIT_VALUE;
    u8 array[I2C_BUFFER_LEN];
    u8 stringpos = BNO055_INIT_VALUE;

    array[BNO055_INIT_VALUE] = reg_addr;

    i2c_start();
    BNO055_iERROR = i2c_write(dev_addr<<1);

    for (stringpos = BNO055_INIT_VALUE; stringpos < (cnt+BNO055_I2C_BUS_WRITE_ARRAY_INDEX); stringpos++)
    {
        BNO055_iERROR = i2c_write(array[stringpos]);
        array[stringpos + BNO055_I2C_BUS_WRITE_ARRAY_INDEX] = *(reg_data + stringpos);
    }

    i2c_stop();


    /*
     * Please take the below APIs as your reference for
     * write the data using I2C communication
     * "BNO055_iERROR = I2C_WRITE_STRING(DEV_ADDR, ARRAY, CNT+1)"
     * add your I2C write APIs here
     * BNO055_iERROR is an return value of I2C read API
     * Please select your valid return value
     * In the driver BNO055_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:
     */

    /*if(BNO055_iERROR)
        BNO055_iERROR = -1;
    else
        BNO055_iERROR = 0;

     return (s8)(BNO055_iERROR);*/
    // Error comm return

    if(BNO055_iERROR-1 != 0)
        BNO055_iERROR = -1;
    else
        BNO055_iERROR = 0;

    return (s8)(BNO055_iERROR);
}

/*  \Brief: The API 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 BNO055_I2C_bus_read(u8 dev_addr, u8 reg_addr, u8 *reg_data, u8 cnt)
{
    s8 BNO055_iERROR = BNO055_INIT_VALUE;
    u8 array[I2C_BUFFER_LEN] = { BNO055_INIT_VALUE };
    u8 stringpos = BNO055_INIT_VALUE;

    array[BNO055_INIT_VALUE] = reg_addr;

    i2c_start();
    // Write asked register
    BNO055_iERROR = i2c_write(dev_addr<<1);
    BNO055_iERROR = i2c_write(reg_addr);
    // Send read address
    i2c_reStart();
    dev_addr = (dev_addr<<1) | 0b00000001;
    BNO055_iERROR = i2c_write(dev_addr);

    /* Please take the below API as your reference
     * for read the data using I2C communication
     * add your I2C read API here.
     * "BNO055_iERROR = I2C_WRITE_READ_STRING(DEV_ADDR,
     * ARRAY, ARRAY, 1, CNT)"
     * BNO055_iERROR is an return value of SPI write API
     * Please select your valid return value
     * In the driver BNO055_SUCCESS defined as 0
     * and FAILURE defined as -1
     */
    for (stringpos = BNO055_INIT_VALUE; stringpos < cnt; stringpos++)
    {

        if(((stringpos+1) < cnt)&&(cnt > BNO055_I2C_BUS_WRITE_ARRAY_INDEX))
            array[stringpos] = i2c_read(1);
        else
            array[stringpos] = i2c_read(0);

        *(reg_data + stringpos) = array[stringpos];

    }

    i2c_stop();

    // Error comm return
    if(BNO055_iERROR-1 != 0)
        BNO055_iERROR = -1;
    else
        BNO055_iERROR = 0;

    return (s8)(BNO055_iERROR);
}

/*  Brief : The delay routine
 *  \param : delay in ms
 */
void BNO055_delay_msek(u32 msek)
{
    /*Delay routine*/
    DRV_TMR0_Stop();
    DRV_TMR0_CounterClear();
    timerData.TmrCnt = 0;
    DRV_TMR0_Start();
    while (timerData.TmrCnt < msek)
    { }
    DRV_TMR0_Stop();
}

#endif


s32 bno055_init_readout(void)
{
    /* Variable used to return value of
     * communication routine*/
    s32 comres = BNO055_ERROR;

    /* variable used to set the power mode of the sensor*/
    u8 power_mode = BNO055_INIT_VALUE;


    /* variable used to read the accel xyz data */
    struct bno055_accel_t accel_xyz;

    /*********read raw mag data***********/
    /* structure used to read the mag xyz data */
    struct bno055_mag_t mag_xyz;

    /***********read raw gyro data***********/
    /* structure used to read the gyro xyz data */
    struct bno055_gyro_t gyro_xyz;

    /*************read raw Euler data************/
    /* structure used to read the euler hrp data */
    struct bno055_euler_t euler_hrp;

    /************read raw quaternion data**************/
    /* structure used to read the quaternion wxyz data */
    struct bno055_quaternion_t quaternion_wxyz;

    /************read raw linear acceleration data***********/
    /* structure used to read the linear accel xyz data */
    struct bno055_linear_accel_t linear_acce_xyz;

    /*****************read raw gravity sensor data****************/
    /* structure used to read the gravity xyz data */
    struct bno055_gravity_t gravity_xyz;

    /*************read accel converted data***************/
    /* structure used to read the accel xyz data output as m/s2 or mg */
    struct bno055_accel_double_t d_accel_xyz;

    /******************read mag converted data********************/
    /* structure used to read the mag xyz data output as uT*/
    struct bno055_mag_double_t d_mag_xyz;

    /*****************read gyro converted data************************/
    /* structure used to read the gyro xyz data output as dps or rps */
    struct bno055_gyro_double_t d_gyro_xyz;

    /*******************read euler converted data*******************/
    /* variable used to read the euler h data output
     * as degree or radians*/
    double d_euler_data_h = BNO055_INIT_VALUE;
    /* variable used to read the euler r data output
     * as degree or radians*/
    double d_euler_data_r = BNO055_INIT_VALUE;
    /* variable used to read the euler p data output
     * as degree or radians*/
    double d_euler_data_p = BNO055_INIT_VALUE;
    /* structure used to read the euler hrp data output
     * as as degree or radians */
    struct bno055_euler_double_t d_euler_hpr;

    /*********read linear acceleration converted data**********/
    /* structure used to read the linear accel xyz data output as m/s2*/
    struct bno055_linear_accel_double_t d_linear_accel_xyz;

    /********************Gravity converted data**********************/
    /* structure used to read the gravity xyz data output as m/s2*/
    struct bno055_gravity_double_t d_gravity_xyz;

    /*---------------------------------------------------------------------------*
     *********************** START INITIALIZATION ************************
     *--------------------------------------------------------------------------*/
#ifdef  BNO055_API

    /*  Based on the user need configure I2C interface.
     *  It is example code to explain how to use the bno055 API*/
    I2C_routine();
#endif

    /*--------------------------------------------------------------------------*
     *  This API used to assign the value/reference of
     *  the following parameters
     *  I2C address
     *  Bus Write
     *  Bus read
     *  Chip id
     *  Page id
     *  Accel revision id
     *  Mag revision id
     *  Gyro revision id
     *  Boot loader revision id
     *  Software revision id
     *-------------------------------------------------------------------------*/
    comres = bno055_init(&bno055);

    /*  For initializing the BNO sensor it is required to the operation mode
     * of the sensor as NORMAL
     * Normal mode can set from the register
     * Page - page0
     * register - 0x3E
     * bit positions - 0 and 1*/
    power_mode = BNO055_POWER_MODE_NORMAL;

    /* set the power mode as NORMAL*/
    comres += bno055_set_power_mode(power_mode);

    /*----------------------------------------------------------------*
     ************************* END INITIALIZATION *************************
     *-----------------------------------------------------------------*/

    /************************* START READ RAW SENSOR DATA****************/

    /*  Using BNO055 sensor we can read the following sensor data and
     * virtual sensor data
     * Sensor data:
     * Accel
     * Mag
     * Gyro
     * Virtual sensor data
     * Euler
     * Quaternion
     * Linear acceleration
     * Gravity sensor */

    /*  For reading sensor raw data it is required to set the
     * operation modes of the sensor
     * operation mode can set from the register
     * page - page0
     * register - 0x3D
     * bit - 0 to 3
     * for sensor data read following operation mode have to set
     * SENSOR MODE
     * 0x01 - BNO055_OPERATION_MODE_ACCONLY
     * 0x02 - BNO055_OPERATION_MODE_MAGONLY
     * 0x03 - BNO055_OPERATION_MODE_GYRONLY
     * 0x04 - BNO055_OPERATION_MODE_ACCMAG
     * 0x05 - BNO055_OPERATION_MODE_ACCGYRO
     * 0x06 - BNO055_OPERATION_MODE_MAGGYRO
     * 0x07 - BNO055_OPERATION_MODE_AMG
     * based on the user need configure the operation mode*/
    comres += bno055_set_operation_mode(BNO055_OPERATION_MODE_AMG);

    /*  Raw accel X, Y and Z data can read from the register
     * page - page 0
     * register - 0x08 to 0x0D*/
    comres += bno055_read_accel_xyz(&accel_xyz);

    /*  Raw mag X, Y and Z data can read from the register
     * page - page 0
     * register - 0x0E to 0x13*/
    comres += bno055_read_mag_xyz(&mag_xyz);

    /*  Raw gyro X, Y and Z data can read from the register
     * page - page 0
     * register - 0x14 to 0x19*/
    comres += bno055_read_gyro_xyz(&gyro_xyz);

    /************************* END READ RAW SENSOR DATA****************/

    /************************* START READ RAW FUSION DATA ********
     * For reading fusion data it is required to set the
     * operation modes of the sensor
     * operation mode can set from the register
     * page - page0
     * register - 0x3D
     * bit - 0 to 3
     * for sensor data read following operation mode have to set
     * FUSION MODE
     * 0x08 - BNO055_OPERATION_MODE_IMUPLUS
     * 0x09 - BNO055_OPERATION_MODE_COMPASS
     * 0x0A - BNO055_OPERATION_MODE_M4G
     * 0x0B - BNO055_OPERATION_MODE_NDOF_FMC_OFF
     * 0x0C - BNO055_OPERATION_MODE_NDOF
     * based on the user need configure the operation mode*/
    comres += bno055_set_operation_mode(BNO055_OPERATION_MODE_NDOF);

    /*  Raw Euler H, R and P data can read from the register
     * page - page 0
     * register - 0x1A to 0x1E */
    //comres += bno055_read_euler_h(&euler_data_h);
    //comres += bno055_read_euler_r(&euler_data_r);
    //comres += bno055_read_euler_p(&euler_data_p);
    comres += bno055_read_euler_hrp(&euler_hrp);

    /*  Raw Quaternion W, X, Y and Z data can read from the register
     * page - page 0
     * register - 0x20 to 0x27 */
    //comres += bno055_read_quaternion_w(&quaternion_data_w);
    //comres += bno055_read_quaternion_x(&quaternion_data_x);
    //comres += bno055_read_quaternion_y(&quaternion_data_y);
    //comres += bno055_read_quaternion_z(&quaternion_data_z);
    comres += bno055_read_quaternion_wxyz(&quaternion_wxyz);

    /*  Raw Linear accel X, Y and Z data can read from the register
     * page - page 0
     * register - 0x28 to 0x2D */
    //comres += bno055_read_linear_accel_x(&linear_accel_data_x);
    //comres += bno055_read_linear_accel_y(&linear_accel_data_y);
    //comres += bno055_read_linear_accel_z(&linear_accel_data_z);
    comres += bno055_read_linear_accel_xyz(&linear_acce_xyz);

    /*  Raw Gravity sensor X, Y and Z data can read from the register
     * page - page 0
     * register - 0x2E to 0x33 */
    //comres += bno055_read_gravity_x(&gravity_data_x);
    //comres += bno055_read_gravity_y(&gravity_data_y);
    //comres += bno055_read_gravity_z(&gravity_data_z);
    comres += bno055_read_gravity_xyz(&gravity_xyz);

    /************************* END READ RAW FUSION DATA  ************/
```

```
/******************START READ CONVERTED SENSOR DATA****************/

/*  API used to read accel data output as double  - m/s2 and mg
 * float functions also available in the BNO055 API */
//comres += bno055_convert_double_accel_x_msq(&d_accel_datax);
//comres += bno055_convert_double_accel_x_mg(&d_accel_datax);
//comres += bno055_convert_double_accel_y_msq(&d_accel_datay);
//comres += bno055_convert_double_accel_y_mg(&d_accel_datay);
//comres += bno055_convert_double_accel_z_msq(&d_accel_dataz);
//comres += bno055_convert_double_accel_z_mg(&d_accel_dataz);
comres += bno055_convert_double_accel_xyz_msq(&d_accel_xyz);
comres += bno055_convert_double_accel_xyz_mg(&d_accel_xyz);

/*  API used to read mag data output as double  - uT(micro Tesla)
 * float functions also available in the BNO055 API */
//comres += bno055_convert_double_mag_x_uT(&d_mag_datax);
//comres += bno055_convert_double_mag_y_uT(&d_mag_datay);
//comres += bno055_convert_double_mag_z_uT(&d_mag_dataz);
comres += bno055_convert_double_mag_xyz_uT(&d_mag_xyz);

/*  API used to read gyro data output as double  - dps and rps
 * float functions also available in the BNO055 API */
//comres += bno055_convert_double_gyro_x_dps(&d_gyro_datax);
//comres += bno055_convert_double_gyro_y_dps(&d_gyro_datay);
//comres += bno055_convert_double_gyro_z_dps(&d_gyro_dataz);
//comres += bno055_convert_double_gyro_x_rps(&d_gyro_datax);
//comres += bno055_convert_double_gyro_y_rps(&d_gyro_datay);
//comres += bno055_convert_double_gyro_z_rps(&d_gyro_dataz);
comres += bno055_convert_double_gyro_xyz_dps(&d_gyro_xyz);
//comres += bno055_convert_double_gyro_xyz_rps(&d_gyro_xyz);

/*  API used to read Euler data output as double  - degree and radians
 * float functions also available in the BNO055 API */
comres += bno055_convert_double_euler_h_deg(&d_euler_data_h);
comres += bno055_convert_double_euler_r_deg(&d_euler_data_r);
comres += bno055_convert_double_euler_p_deg(&d_euler_data_p);
//comres += bno055_convert_double_euler_h_rad(&d_euler_data_h);
//comres += bno055_convert_double_euler_r_rad(&d_euler_data_r);
//comres += bno055_convert_double_euler_p_rad(&d_euler_data_p);
comres += bno055_convert_double_euler_hpr_deg(&d_euler_hpr);
//comres += bno055_convert_double_euler_hpr_rad(&d_euler_hpr);

/*  API used to read Linear acceleration data output as m/s2
 * float functions also available in the BNO055 API */
//comres += bno055_convert_double_linear_accel_x_msq(&d_linear_accel_datax);
//comres += bno055_convert_double_linear_accel_y_msq(&d_linear_accel_datay);
//comres += bno055_convert_double_linear_accel_z_msq(&d_linear_accel_dataz);
comres += bno055_convert_double_linear_accel_xyz_msq(&d_linear_accel_xyz);

/*  API used to read Gravity sensor data output as m/s2
 * float functions also available in the BNO055 API */
//comres += bno055_convert_gravity_double_x_msq(&d_gravity_data_x);
//comres += bno055_convert_gravity_double_y_msq(&d_gravity_data_y);
//comres += bno055_convert_gravity_double_z_msq(&d_gravity_data_z);
comres += bno055_convert_double_gravity_xyz_msq(&d_gravity_xyz);

/*--------------------------------------------------------------------*
 ************************* START DE-INITIALIZATION ***********************
 *--------------------------------------------------------------------*/

/*  For de - initializing the BNO sensor it is required
 * to the operation mode of the sensor as SUSPEND
 * Suspend mode can set from the register
 * Page - page0
 * register - 0x3E
 * bit positions - 0 and 1*/
//power_mode = BNO055_POWER_MODE_SUSPEND;

/* set the power mode as SUSPEND*/
//comres += bno055_set_power_mode(power_mode);
comres += bno055_set_operation_mode(BNO055_OPERATION_MODE_NDOF);
/*------------------------------------------------------------------*
************************* END DE-INITIALIZATION **********************
*------------------------------------------------------------------*/
return comres;
```

```
}

```