```c
/**
* @file bno055_support.c
*
*/

/*-------------------------------------------------------------------------*
*  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 timeData;

#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
```

```
74          * float functions also available in the BNO055 API */
75         comres += bno055_convert_double_euler_hpr_deg(&data->euler);
76         /*  API used to read Linear acceleration data output as m/s2
77          * float functions also available in the BNO055 API */
78         comres += bno055_convert_double_linear_accel_xyz_msq(&data->linear_accel);
79         comres += bno055_convert_double_gravity_xyz_msq(&data->gravity);
80
81         /*---------------------------------------------------------------------------*
82          ************************ START DE-INITIALIZATION ***********************
83          *---------------------------------------------------------------------------*/
84
85         /*  For de - initializing the BNO sensor it is required
86          * to the operation mode of the sensor as SUSPEND
87          * Suspend mode can set from the register
88          * Page - page0
89          * register - 0x3E
90          * bit positions - 0 and 1*/
91         //power_mode = BNO055_POWER_MODE_SUSPEND;
92
93         /* set the power mode as SUSPEND*/
94         //comres += bno055_set_power_mode(power_mode);
95
96         /* Flag measure ready */
97         data->flagMeasReady = true;
98
99         /*---------------------------------------------------------------------------*
100        ************************ END DE-INITIALIZATION **********************
101        *---------------------------------------------------------------------------*/
102        return (comres+1);
103    }
104
105    /*---------------------------------------------------------------------------*
106     *  The following API is used to map the I2C bus read, write, delay and
107     *  device address with global structure bno055_t
108     *---------------------------------------------------------------------------*/
109
110    /*---------------------------------------------------------------------------*
111     *  By using bno055 the following structure parameter can be accessed
112     *  Bus write function pointer: BNO055_WR_FUNC_PTR
113     *  Bus read function pointer: BNO055_RD_FUNC_PTR
114     *  Delay function pointer: delay_msec
115     *  I2C address: dev_addr
116     *---------------------------------------------------------------------------*/
117    s8 I2C_routine(void)
118    {
119        bno055.bus_write = BNO055_I2C_bus_write;
120        bno055.bus_read = BNO055_I2C_bus_read;
121        bno055.delay_msec = BNO055_delay_msek;
122        bno055.dev_addr = BNO055_I2C_ADDR1;
123        return BNO055_INIT_VALUE;
124    }
125
126    /************** I2C buffer length******/
127
128    #define I2C_BUFFER_LEN 8
129    #define I2C0           5
130
131    /*---------------------------------------------------------------------------*
132     *
133     *  This is a sample code for read and write the data by using I2C
134     *  Use either I2C  based on your need
135     *  The device address defined in the bno055.h file
136     *
137     *---------------------------------------------------------------------------*/
138
139    /*  \Brief: The API is used as I2C bus write
140     *  \Return : Status of the I2C write
141     *  \param dev_addr : The device address of the sensor
142     *  \param reg_addr : Address of the first register,
143     *   will data is going to be written
144     *  \param reg_data : It is a value hold in the array,
145     *      will be used for write the value into the register
146     *  \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();
```

```
219         // Write asked register
220         BNO055_iERROR = i2c_write(dev_addr<<1);
221         BNO055_iERROR = i2c_write(reg_addr);
222         // Send read address
223         i2c_reStart();
224         dev_addr = (dev_addr<<1) | 0b00000001;
225         BNO055_iERROR = i2c_write(dev_addr);
226
227         /* Please take the below API as your reference
228          * for read the data using I2C communication
229          * add your I2C read API here.
230          * "BNO055_iERROR = I2C_WRITE_READ_STRING(DEV_ADDR,
231          * ARRAY, ARRAY, 1, CNT)"
232          * BNO055_iERROR is an return value of SPI write API
233          * Please select your valid return value
234          * In the driver BNO055_SUCCESS defined as 0
235          * and FAILURE defined as -1
236          */
237         for (stringpos = BNO055_INIT_VALUE; stringpos < cnt; stringpos++)
238         {
239
240             if(((stringpos+1) < cnt)&&(cnt > BNO055_I2C_BUS_WRITE_ARRAY_INDEX))
241                 array[stringpos] = i2c_read(1);
242             else
243                 array[stringpos] = i2c_read(0);
244
245             *(reg_data + stringpos) = array[stringpos];
246
247         }
248
249         i2c_stop();
250
251         // Error comm return
252         if(BNO055_iERROR-1 != 0)
253             BNO055_iERROR = -1;
254         else
255             BNO055_iERROR = 0;
256
257         return (s8)(BNO055_iERROR);
258     }
259
260     /*  Brief : The delay routine
261      *  \param : delay in ms
262      */
263     void BNO055_delay_msek(u32 msek)
264     {
265         /*Delay routine*/
266         DRV_TMR0_Stop();
267         DRV_TMR0_CounterClear();
268         timeData.delayCnt = 0;
269         DRV_TMR0_Start();
270         while (timeData.delayCnt < msek)
271         { }
272         DRV_TMR0_Stop();
273     }
274
275     #endif
276
277
278     s32 bno055_init_readout(void)
279     {
280         /* Variable used to return value of
281          * communication routine*/
282         s32 comres = BNO055_ERROR;
283
284         /* variable used to set the power mode of the sensor*/
285         u8 power_mode = BNO055_INIT_VALUE;
286
287
288         /* variable used to read the accel xyz data */
289         struct bno055_accel_t accel_xyz;
290
291         /*********read raw mag data**********/
```

```
292      /* structure used to read the mag xyz data */
293      struct bno055_mag_t mag_xyz;
294
295      /***********read raw gyro data***********/
296      /* structure used to read the gyro xyz data */
297      struct bno055_gyro_t gyro_xyz;
298
299      /************read raw Euler data************/
300      /* structure used to read the euler hrp data */
301      struct bno055_euler_t euler_hrp;
302
303      /************read raw quaternion data*************/
304      /* structure used to read the quaternion wxyz data */
305      struct bno055_quaternion_t quaternion_wxyz;
306
307      /************read raw linear acceleration data***********/
308      /* structure used to read the linear accel xyz data */
309      struct bno055_linear_accel_t linear_acce_xyz;
310
311      /*****************read raw gravity sensor data***************/
312      /* structure used to read the gravity xyz data */
313      struct bno055_gravity_t gravity_xyz;
314
315      /************read accel converted data**************/
316      /* structure used to read the accel xyz data output as m/s2 or mg */
317      struct bno055_accel_double_t d_accel_xyz;
318
319      /******************read mag converted data*******************/
320      /* structure used to read the mag xyz data output as uT*/
321      struct bno055_mag_double_t d_mag_xyz;
322
323      /*****************read gyro converted data************************/
324      /* structure used to read the gyro xyz data output as dps or rps */
325      struct bno055_gyro_double_t d_gyro_xyz;
326
327      /*******************read euler converted data******************/
328      /* variable used to read the euler h data output
329       * as degree or radians*/
330      double d_euler_data_h = BNO055_INIT_VALUE;
331      /* variable used to read the euler r data output
332       * as degree or radians*/
333      double d_euler_data_r = BNO055_INIT_VALUE;
334      /* variable used to read the euler p data output
335       * as degree or radians*/
336      double d_euler_data_p = BNO055_INIT_VALUE;
337      /* structure used to read the euler hrp data output
338       * as as degree or radians */
339      struct bno055_euler_double_t d_euler_hpr;
340
341      /*********read linear acceleration converted data**********/
342      /* structure used to read the linear accel xyz data output as m/s2*/
343      struct bno055_linear_accel_double_t d_linear_accel_xyz;
344
345      /*******************Gravity converted data*********************/
346      /* structure used to read the gravity xyz data output as m/s2*/
347      struct bno055_gravity_double_t d_gravity_xyz;
348
349      /*---------------------------------------------------------------------------*
350       ********************** START INITIALIZATION ************************
351       *---------------------------------------------------------------------------*/
352  #ifdef  BNO055_API
353
354      /*  Based on the user need configure I2C interface.
355       *  It is example code to explain how to use the bno055 API*/
356      I2C_routine();
357  #endif
358
359      /*---------------------------------------------------------------------------*
360       *  This API used to assign the value/reference of
361       *  the following parameters
362       *  I2C address
363       *  Bus Write
364       *  Bus read
```

```
365         *  Chip id
366         *  Page id
367         *  Accel revision id
368         *  Mag revision id
369         *  Gyro revision id
370         *  Boot loader revision id
371         *  Software revision id
372         *-------------------------------------------------------------------------*/
373        comres = bno055_init(&bno055);
374
375        /*  For initializing the BNO sensor it is required to the operation mode
376         * of the sensor as NORMAL
377         * Normal mode can set from the register
378         * Page - page0
379         * register - 0x3E
380         * bit positions - 0 and 1*/
381        power_mode = BNO055_POWER_MODE_NORMAL;
382
383        /* set the power mode as NORMAL*/
384        comres += bno055_set_power_mode(power_mode);
385
386        /*-----------------------------------------------------------------*
387         ************************ END INITIALIZATION ************************
388         *-----------------------------------------------------------------*/
389
390        /************************ START READ RAW SENSOR DATA****************/
391
392        /*  Using BNO055 sensor we can read the following sensor data and
393         * virtual sensor data
394         * Sensor data:
395         * Accel
396         * Mag
397         * Gyro
398         * Virtual sensor data
399         * Euler
400         * Quaternion
401         * Linear acceleration
402         * Gravity sensor */
403
404        /*  For reading sensor raw data it is required to set the
405         * operation modes of the sensor
406         * operation mode can set from the register
407         * page - page0
408         * register - 0x3D
409         * bit - 0 to 3
410         * for sensor data read following operation mode have to set
411         * SENSOR MODE
412         * 0x01 - BNO055_OPERATION_MODE_ACCONLY
413         * 0x02 - BNO055_OPERATION_MODE_MAGONLY
414         * 0x03 - BNO055_OPERATION_MODE_GYRONLY
415         * 0x04 - BNO055_OPERATION_MODE_ACCMAG
416         * 0x05 - BNO055_OPERATION_MODE_ACCGYRO
417         * 0x06 - BNO055_OPERATION_MODE_MAGGYRO
418         * 0x07 - BNO055_OPERATION_MODE_AMG
419         * based on the user need configure the operation mode*/
420        comres += bno055_set_operation_mode(BNO055_OPERATION_MODE_AMG);
421
422        /*  Raw accel X, Y and Z data can read from the register
423         * page - page 0
424         * register - 0x08 to 0x0D*/
425        comres += bno055_read_accel_xyz(&accel_xyz);
426
427        /*  Raw mag X, Y and Z data can read from the register
428         * page - page 0
429         * register - 0x0E to 0x13*/
430        comres += bno055_read_mag_xyz(&mag_xyz);
431
432        /*  Raw gyro X, Y and Z data can read from the register
433         * page - page 0
434         * register - 0x14 to 0x19*/
435        comres += bno055_read_gyro_xyz(&gyro_xyz);
436
437        /************************ END READ RAW SENSOR DATA****************/
```

```
438
439         /************************ START READ RAW FUSION DATA ********
440          * For reading fusion data it is required to set the
441          * operation modes of the sensor
442          * operation mode can set from the register
443          * page - page0
444          * register - 0x3D
445          * bit - 0 to 3
446          * for sensor data read following operation mode have to set
447          * FUSION MODE
448          * 0x08 - BNO055_OPERATION_MODE_IMUPLUS
449          * 0x09 - BNO055_OPERATION_MODE_COMPASS
450          * 0x0A - BNO055_OPERATION_MODE_M4G
451          * 0x0B - BNO055_OPERATION_MODE_NDOF_FMC_OFF
452          * 0x0C - BNO055_OPERATION_MODE_NDOF
453          * based on the user need configure the operation mode*/
454         comres += bno055_set_operation_mode(BNO055_OPERATION_MODE_NDOF);
455
456         /*  Raw Euler H, R and P data can read from the register
457          * page - page 0
458          * register - 0x1A to 0x1E */
459         //comres += bno055_read_euler_h(&euler_data_h);
460         //comres += bno055_read_euler_r(&euler_data_r);
461         //comres += bno055_read_euler_p(&euler_data_p);
462         comres += bno055_read_euler_hrp(&euler_hrp);
463
464         /*  Raw Quaternion W, X, Y and Z data can read from the register
465          * page - page 0
466          * register - 0x20 to 0x27 */
467         //comres += bno055_read_quaternion_w(&quaternion_data_w);
468         //comres += bno055_read_quaternion_x(&quaternion_data_x);
469         //comres += bno055_read_quaternion_y(&quaternion_data_y);
470         //comres += bno055_read_quaternion_z(&quaternion_data_z);
471         comres += bno055_read_quaternion_wxyz(&quaternion_wxyz);
472
473         /*  Raw Linear accel X, Y and Z data can read from the register
474          * page - page 0
475          * register - 0x28 to 0x2D */
476         //comres += bno055_read_linear_accel_x(&linear_accel_data_x);
477         //comres += bno055_read_linear_accel_y(&linear_accel_data_y);
478         //comres += bno055_read_linear_accel_z(&linear_accel_data_z);
479         comres += bno055_read_linear_accel_xyz(&linear_acce_xyz);
480
481         /*  Raw Gravity sensor X, Y and Z data can read from the register
482          * page - page 0
483          * register - 0x2E to 0x33 */
484         //comres += bno055_read_gravity_x(&gravity_data_x);
485         //comres += bno055_read_gravity_y(&gravity_data_y);
486         //comres += bno055_read_gravity_z(&gravity_data_z);
487         comres += bno055_read_gravity_xyz(&gravity_xyz);
488
489         /************************ END READ RAW FUSION DATA  ************/
490         /*****************START READ CONVERTED SENSOR DATA****************/
491
492         /*  API used to read accel data output as double  - m/s2 and mg
493          * float functions also available in the BNO055 API */
494         //comres += bno055_convert_double_accel_x_msq(&d_accel_datax);
495         //comres += bno055_convert_double_accel_x_mg(&d_accel_datax);
496         //comres += bno055_convert_double_accel_y_msq(&d_accel_datay);
497         //comres += bno055_convert_double_accel_y_mg(&d_accel_datay);
498         //comres += bno055_convert_double_accel_z_msq(&d_accel_dataz);
499         //comres += bno055_convert_double_accel_z_mg(&d_accel_dataz);
500         comres += bno055_convert_double_accel_xyz_msq(&d_accel_xyz);
501         comres += bno055_convert_double_accel_xyz_mg(&d_accel_xyz);
502
503         /*  API used to read mag data output as double  - uT(micro Tesla)
504          * float functions also available in the BNO055 API */
505         //comres += bno055_convert_double_mag_x_uT(&d_mag_datax);
506         //comres += bno055_convert_double_mag_y_uT(&d_mag_datay);
507         //comres += bno055_convert_double_mag_z_uT(&d_mag_dataz);
508         comres += bno055_convert_double_mag_xyz_uT(&d_mag_xyz);
509
510         /*  API used to read gyro data output as double  - dps and rps
```

```
511         * float functions also available in the BNO055 API */
512        //comres += bno055_convert_double_gyro_x_dps(&d_gyro_datax);
513        //comres += bno055_convert_double_gyro_y_dps(&d_gyro_datay);
514        //comres += bno055_convert_double_gyro_z_dps(&d_gyro_dataz);
515        //comres += bno055_convert_double_gyro_x_rps(&d_gyro_datax);
516        //comres += bno055_convert_double_gyro_y_rps(&d_gyro_datay);
517        //comres += bno055_convert_double_gyro_z_rps(&d_gyro_dataz);
518        comres += bno055_convert_double_gyro_xyz_dps(&d_gyro_xyz);
519        //comres += bno055_convert_double_gyro_xyz_rps(&d_gyro_xyz);
520
521        /*  API used to read Euler data output as double  - degree and radians
522         * float functions also available in the BNO055 API */
523        comres += bno055_convert_double_euler_h_deg(&d_euler_data_h);
524        comres += bno055_convert_double_euler_r_deg(&d_euler_data_r);
525        comres += bno055_convert_double_euler_p_deg(&d_euler_data_p);
526        //comres += bno055_convert_double_euler_h_rad(&d_euler_data_h);
527        //comres += bno055_convert_double_euler_r_rad(&d_euler_data_r);
528        //comres += bno055_convert_double_euler_p_rad(&d_euler_data_p);
529        comres += bno055_convert_double_euler_hpr_deg(&d_euler_hpr);
530        //comres += bno055_convert_double_euler_hpr_rad(&d_euler_hpr);
531
532        /*  API used to read Linear acceleration data output as m/s2
533         * float functions also available in the BNO055 API */
534        //comres += bno055_convert_double_linear_accel_x_msq(&d_linear_accel_datax);
535        //comres += bno055_convert_double_linear_accel_y_msq(&d_linear_accel_datay);
536        //comres += bno055_convert_double_linear_accel_z_msq(&d_linear_accel_dataz);
537        comres += bno055_convert_double_linear_accel_xyz_msq(&d_linear_accel_xyz);
538
539        /*  API used to read Gravity sensor data output as m/s2
540         * float functions also available in the BNO055 API */
541        //comres += bno055_convert_gravity_double_x_msq(&d_gravity_data_x);
542        //comres += bno055_convert_gravity_double_y_msq(&d_gravity_data_y);
543        //comres += bno055_convert_gravity_double_z_msq(&d_gravity_data_z);
544        comres += bno055_convert_double_gravity_xyz_msq(&d_gravity_xyz);
545
546        /*---------------------------------------------------------------------*
547         ************************ START DE-INITIALIZATION ***********************
548         *---------------------------------------------------------------------*/
549
550        /*  For de - initializing the BNO sensor it is required
551         * to the operation mode of the sensor as SUSPEND
552         * Suspend mode can set from the register
553         * Page - page0
554         * register - 0x3E
555         * bit positions - 0 and 1*/
556        //power_mode = BNO055_POWER_MODE_SUSPEND;
557
558        /* set the power mode as SUSPEND*/
559        //comres += bno055_set_power_mode(power_mode);
560        comres += bno055_set_operation_mode(BNO055_OPERATION_MODE_NDOF);
561        /*---------------------------------------------------------------------*
562        ************************ END DE-INITIALIZATION **********************
563        *---------------------------------------------------------------------*/
564        return comres;
565    }
566
```