```
/**
* 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;
```

```
223     return (s8)(BNO055_iERROR);*/
224     // Error comm return
225
226     if(BNO055_iERROR-1 != 0)
227         BNO055_iERROR = -1;
228     else
229         BNO055_iERROR = 0;
230
231     return (s8)(BNO055_iERROR);
232 }
233
234 /*  \Brief: The API is used as I2C bus read
235  *  \Return : Status of the I2C read
236  *  \param dev_addr : The device address of the sensor
237  *  \param reg_addr : Address of the first register,
238  *  will data is going to be read
239  *  \param reg_data : This data read from the sensor,
240  *   which is hold in an array
241  *  \param cnt : The no of byte of data to be read
242  */
243 s8 BNO055_I2C_bus_read(u8 dev_addr, u8 reg_addr, u8 *reg_data, u8 cnt)
244 {
245     s8 BNO055_iERROR = BNO055_INIT_VALUE;
246     u8 array[I2C_BUFFER_LEN] = { BNO055_INIT_VALUE };
247     u8 stringpos = BNO055_INIT_VALUE;
248
249     array[BNO055_INIT_VALUE] = reg_addr;
250
251     i2c_start();
252     // Write asked register
253     BNO055_iERROR = i2c_write(dev_addr<<1);
254     BNO055_iERROR = i2c_write(reg_addr);
255     // Send read address
256     i2c_reStart();
257     dev_addr = (dev_addr<<1) | 0b00000001;
258     BNO055_iERROR = i2c_write(dev_addr);
259
260     /* Please take the below API as your reference
261      * for read the data using I2C communication
262      * add your I2C read API here.
263      * "BNO055_iERROR = I2C_WRITE_READ_STRING(DEV_ADDR,
264      * ARRAY, ARRAY, 1, CNT)"
265      * BNO055_iERROR is an return value of SPI write API
266      * Please select your valid return value
267      * In the driver BNO055_SUCCESS defined as 0
268      * and FAILURE defined as -1
269      */
270     for (stringpos = BNO055_INIT_VALUE; stringpos < cnt; stringpos++)
271     {
272
273         if(((stringpos+1) < cnt)&&(cnt > BNO055_I2C_BUS_WRITE_ARRAY_INDEX))
274             array[stringpos] = i2c_read(1);
275         else
276             array[stringpos] = i2c_read(0);
277
278         *(reg_data + stringpos) = array[stringpos];
279
280     }
281
282     i2c_stop();
283
284     // Error comm return
285     if(BNO055_iERROR-1 != 0)
286         BNO055_iERROR = -1;
287     else
288         BNO055_iERROR = 0;
289
290     return (s8)(BNO055_iERROR);
291 }
292
293 /*  Brief : The delay routine
294  *  \param : delay in ms
295  */
296 void BNO055_delay_msek(u32 msek)
297 {
```

```
    /*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;
```

```
373
374     /*********read linear acceleration converted data**********/
375     /* structure used to read the linear accel xyz data output as m/s2*/
376     struct bno055_linear_accel_double_t d_linear_accel_xyz;
377
378     /*******************Gravity converted data*********************/
379     /* structure used to read the gravity xyz data output as m/s2*/
380     struct bno055_gravity_double_t d_gravity_xyz;
381
382     /*---------------------------------------------------------------------*
383      ********************** START INITIALIZATION ************************
384      *---------------------------------------------------------------------*/
385 #ifdef  BNO055_API
386
387     /*  Based on the user need configure I2C interface.
388      *  It is example code to explain how to use the bno055 API*/
389     I2C_routine();
390 #endif
391
392     /*---------------------------------------------------------------------*
393      *  This API used to assign the value/reference of
394      *  the following parameters
395      *  I2C address
396      *  Bus Write
397      *  Bus read
398      *  Chip id
399      *  Page id
400      *  Accel revision id
401      *  Mag revision id
402      *  Gyro revision id
403      *  Boot loader revision id
404      *  Software revision id
405      *---------------------------------------------------------------------*/
406     comres = bno055_init(&bno055);
407
408     /*  For initializing the BNO sensor it is required to the operation mode
409      * of the sensor as NORMAL
410      * Normal mode can set from the register
411      * Page - page0
412      * register - 0x3E
413      * bit positions - 0 and 1*/
414     power_mode = BNO055_POWER_MODE_NORMAL;
415
416     /* set the power mode as NORMAL*/
417     comres += bno055_set_power_mode(power_mode);
418
419     /*---------------------------------------------------------------*
420      ************************* END INITIALIZATION *************************
421      *---------------------------------------------------------------*/
422
423     /************************* START READ RAW SENSOR DATA****************/
424
425     /*  Using BNO055 sensor we can read the following sensor data and
426      * virtual sensor data
427      * Sensor data:
428      * Accel
429      * Mag
430      * Gyro
431      * Virtual sensor data
432      * Euler
433      * Quaternion
434      * Linear acceleration
435      * Gravity sensor */
436
437     /*  For reading sensor raw data it is required to set the
438      * operation modes of the sensor
439      * operation mode can set from the register
440      * page - page0
441      * register - 0x3D
442      * bit - 0 to 3
443      * for sensor data read following operation mode have to set
444      * SENSOR MODE
445      * 0x01 - BNO055_OPERATION_MODE_ACCONLY
446      * 0x02 - BNO055_OPERATION_MODE_MAGONLY
447      * 0x03 - BNO055_OPERATION_MODE_GYRONLY
```

```
448     * 0x04 - BNO055_OPERATION_MODE_ACCMAG
449     * 0x05 - BNO055_OPERATION_MODE_ACCGYRO
450     * 0x06 - BNO055_OPERATION_MODE_MAGGYRO
451     * 0x07 - BNO055_OPERATION_MODE_AMG
452     * based on the user need configure the operation mode*/
453    comres += bno055_set_operation_mode(BNO055_OPERATION_MODE_AMG);
454
455    /*  Raw accel X, Y and Z data can read from the register
456     * page - page 0
457     * register - 0x08 to 0x0D*/
458    comres += bno055_read_accel_xyz(&accel_xyz);
459
460    /*  Raw mag X, Y and Z data can read from the register
461     * page - page 0
462     * register - 0x0E to 0x13*/
463    comres += bno055_read_mag_xyz(&mag_xyz);
464
465    /*  Raw gyro X, Y and Z data can read from the register
466     * page - page 0
467     * register - 0x14 to 0x19*/
468    comres += bno055_read_gyro_xyz(&gyro_xyz);
469
470    /************************* END READ RAW SENSOR DATA****************/
471
472    /************************* START READ RAW FUSION DATA ********
473     * For reading fusion data it is required to set the
474     * operation modes of the sensor
475     * operation mode can set from the register
476     * page - page0
477     * register - 0x3D
478     * bit - 0 to 3
479     * for sensor data read following operation mode have to set
480     * FUSION MODE
481     * 0x08 - BNO055_OPERATION_MODE_IMUPLUS
482     * 0x09 - BNO055_OPERATION_MODE_COMPASS
483     * 0x0A - BNO055_OPERATION_MODE_M4G
484     * 0x0B - BNO055_OPERATION_MODE_NDOF_FMC_OFF
485     * 0x0C - BNO055_OPERATION_MODE_NDOF
486     * based on the user need configure the operation mode*/
487    comres += bno055_set_operation_mode(BNO055_OPERATION_MODE_NDOF);
488
489    /*  Raw Euler H, R and P data can read from the register
490     * page - page 0
491     * register - 0x1A to 0x1E */
492    //comres += bno055_read_euler_h(&euler_data_h);
493    //comres += bno055_read_euler_r(&euler_data_r);
494    //comres += bno055_read_euler_p(&euler_data_p);
495    comres += bno055_read_euler_hrp(&euler_hrp);
496
497    /*  Raw Quaternion W, X, Y and Z data can read from the register
498     * page - page 0
499     * register - 0x20 to 0x27 */
500    //comres += bno055_read_quaternion_w(&quaternion_data_w);
501    //comres += bno055_read_quaternion_x(&quaternion_data_x);
502    //comres += bno055_read_quaternion_y(&quaternion_data_y);
503    //comres += bno055_read_quaternion_z(&quaternion_data_z);
504    comres += bno055_read_quaternion_wxyz(&quaternion_wxyz);
505
506    /*  Raw Linear accel X, Y and Z data can read from the register
507     * page - page 0
508     * register - 0x28 to 0x2D */
509    //comres += bno055_read_linear_accel_x(&linear_accel_data_x);
510    //comres += bno055_read_linear_accel_y(&linear_accel_data_y);
511    //comres += bno055_read_linear_accel_z(&linear_accel_data_z);
512    comres += bno055_read_linear_accel_xyz(&linear_acce_xyz);
513
514    /*  Raw Gravity sensor X, Y and Z data can read from the register
515     * page - page 0
516     * register - 0x2E to 0x33 */
517    //comres += bno055_read_gravity_x(&gravity_data_x);
518    //comres += bno055_read_gravity_y(&gravity_data_y);
519    //comres += bno055_read_gravity_z(&gravity_data_z);
520    comres += bno055_read_gravity_xyz(&gravity_xyz);
521
522    /************************* END READ RAW FUSION DATA  ************/
```

```
523   /*****************START READ CONVERTED SENSOR DATA****************/
524
525   /*  API used to read accel data output as double  - m/s2 and mg
526    * float functions also available in the BNO055 API */
527   //comres += bno055_convert_double_accel_x_msq(&d_accel_datax);
528   //comres += bno055_convert_double_accel_x_mg(&d_accel_datax);
529   //comres += bno055_convert_double_accel_y_msq(&d_accel_datay);
530   //comres += bno055_convert_double_accel_y_mg(&d_accel_datay);
531   //comres += bno055_convert_double_accel_z_msq(&d_accel_dataz);
532   //comres += bno055_convert_double_accel_z_mg(&d_accel_dataz);
533   comres += bno055_convert_double_accel_xyz_msq(&d_accel_xyz);
534   comres += bno055_convert_double_accel_xyz_mg(&d_accel_xyz);
535
536   /*  API used to read mag data output as double  - uT(micro Tesla)
537    * float functions also available in the BNO055 API */
538   //comres += bno055_convert_double_mag_x_uT(&d_mag_datax);
539   //comres += bno055_convert_double_mag_y_uT(&d_mag_datay);
540   //comres += bno055_convert_double_mag_z_uT(&d_mag_dataz);
541   comres += bno055_convert_double_mag_xyz_uT(&d_mag_xyz);
542
543   /*  API used to read gyro data output as double  - dps and rps
544    * float functions also available in the BNO055 API */
545   //comres += bno055_convert_double_gyro_x_dps(&d_gyro_datax);
546   //comres += bno055_convert_double_gyro_y_dps(&d_gyro_datay);
547   //comres += bno055_convert_double_gyro_z_dps(&d_gyro_dataz);
548   //comres += bno055_convert_double_gyro_x_rps(&d_gyro_datax);
549   //comres += bno055_convert_double_gyro_y_rps(&d_gyro_datay);
550   //comres += bno055_convert_double_gyro_z_rps(&d_gyro_dataz);
551   comres += bno055_convert_double_gyro_xyz_dps(&d_gyro_xyz);
552   //comres += bno055_convert_double_gyro_xyz_rps(&d_gyro_xyz);
553
554   /*  API used to read Euler data output as double  - degree and radians
555    * float functions also available in the BNO055 API */
556   comres += bno055_convert_double_euler_h_deg(&d_euler_data_h);
557   comres += bno055_convert_double_euler_r_deg(&d_euler_data_r);
558   comres += bno055_convert_double_euler_p_deg(&d_euler_data_p);
559   //comres += bno055_convert_double_euler_h_rad(&d_euler_data_h);
560   //comres += bno055_convert_double_euler_r_rad(&d_euler_data_r);
561   //comres += bno055_convert_double_euler_p_rad(&d_euler_data_p);
562   comres += bno055_convert_double_euler_hpr_deg(&d_euler_hpr);
563   //comres += bno055_convert_double_euler_hpr_rad(&d_euler_hpr);
564
565   /*  API used to read Linear acceleration data output as m/s2
566    * float functions also available in the BNO055 API */
567   //comres += bno055_convert_double_linear_accel_x_msq(&d_linear_accel_datax);
568   //comres += bno055_convert_double_linear_accel_y_msq(&d_linear_accel_datay);
569   //comres += bno055_convert_double_linear_accel_z_msq(&d_linear_accel_dataz);
570   comres += bno055_convert_double_linear_accel_xyz_msq(&d_linear_accel_xyz);
571
572   /*  API used to read Gravity sensor data output as m/s2
573    * float functions also available in the BNO055 API */
574   //comres += bno055_convert_gravity_double_x_msq(&d_gravity_data_x);
575   //comres += bno055_convert_gravity_double_y_msq(&d_gravity_data_y);
576   //comres += bno055_convert_gravity_double_z_msq(&d_gravity_data_z);
577   comres += bno055_convert_double_gravity_xyz_msq(&d_gravity_xyz);
578
579   /*---------------------------------------------------------------------*
580    ************************ START DE-INITIALIZATION ***********************
581    *---------------------------------------------------------------------*/
582
583   /*  For de - initializing the BNO sensor it is required
584    * to the operation mode of the sensor as SUSPEND
585    * Suspend mode can set from the register
586    * Page - page0
587    * register - 0x3E
588    * bit positions - 0 and 1*/
589   //power_mode = BNO055_POWER_MODE_SUSPEND;
590
591   /* set the power mode as SUSPEND*/
592   //comres += bno055_set_power_mode(power_mode);
593   comres += bno055_set_operation_mode(BNO055_OPERATION_MODE_NDOF);
594   /*---------------------------------------------------------------------*
595   ************************ END DE-INITIALIZATION **********************
596   *---------------------------------------------------------------------*/
597   return comres;
```

```
598 }
599
```