```
/*******************************************************************************
  MPLAB Harmony Application Source File

  Company:
    Microchip Technology Inc.

  File Name:
    app.c

  Summary:
    This file contains the source code for the MPLAB Harmony application.

  Description:
    This file contains the source code for the MPLAB Harmony application.  It
    implements the logic of the application's state machine and it may call
    API routines of other MPLAB Harmony modules in the system, such as drivers,
    system services, and middleware.  However, it does not call any of the
    system interfaces (such as the "Initialize" and "Tasks" functions) of any of
    the modules in the system or make any assumptions about when those functions
    are called.  That is the responsibility of the configuration-specific system
    files.
 *******************************************************************************/

// DOM-IGNORE-BEGIN
/*******************************************************************************
Copyright (c) 2013-2014 released Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
 *******************************************************************************/
// DOM-IGNORE-END


// *****************************************************************************
// *****************************************************************************
// Section: Included Files
// *****************************************************************************
// *****************************************************************************

#include "app.h"
#include "bno055.h"
#include "bno055_support.h"
#include "GNSS/u_gnss_pos.h"
#include "Mc32_I2cUtilCCS.h"
#include "Mc32_serComm.h"
#include "Mc32_sdFatGest.h"
#include "Mc32Debounce.h"
#include "usart_FIFO.h"
#include "GNSS/u_ubx_protocol.h"
#include <stdio.h>

// *****************************************************************************
// *****************************************************************************
// Section: Global Data Definitions
// *****************************************************************************
// *****************************************************************************
/* Switch descriptor */
```

```
74  S_SwitchDescriptor switchDescr;
75  // *****************************************************************************
76  /* Application Data
77
78    Summary:
79      Holds application data
80
81    Description:
82      This structure holds the application's data.
83
84    Remarks:
85      This structure should be initialized by the APP_Initialize function.
86
87      Application strings and buffers are be defined outside this structure.
88  */
89
90  APP_DATA appData;
91  TIMER_DATA timeData;
92
93
94  // *****************************************************************************
95  // *****************************************************************************
96  // Section: Application Callback Functions
97  // *****************************************************************************
98  // *****************************************************************************
99
100 void delayTimer_callback(){
101     /* Increment delay timer */
102     timeData.delayCnt ++;
103 }
104
105 void stateTimer_callback()
106 {
107     /* Increment all counters */
108     timeData.ledCnt ++;
109     timeData.measCnt[BNO055_idx] ++;
110     timeData.measCnt[GNSS_idx] ++;
111     timeData.inactiveCnt ++;
112     timeData.tmrTickFlag = true;
113     /* When the button is pressed, the hold time is counted. */
114     if(timeData.flagCntBtnPressed){
115         timeData.cntBtnPressed++;
116     }
117      /* Do debounce on button every 10 ms */
118      DoDebounce(&switchDescr, ButtonMFStateGet());
119     /* Start a measure set each IMU period */
120     if ( ( timeData.measCnt[BNO055_idx] % (timeData.measPeriod[BNO055_idx]/10) ) == 0)
121         timeData.measTodo[BNO055_idx] = true;
122
123     /* Start a measure set each GNSS period */
124     if ( ( timeData.measCnt[GNSS_idx] % (timeData.measPeriod[GNSS_idx]/10) ) == 0)
125         timeData.measTodo[GNSS_idx] = true;
126     /* Manage LED if enabled */
127     if((timeData.ledCnt % LED_PERIOD == 0) && (appData.ledState == true))
128        LED_BOff();
129 }
130
131 // *****************************************************************************
132 // *****************************************************************************
133 // Section: Application Local Functions
134 // *****************************************************************************
135 // *****************************************************************************
136 static void stopLogging (void);
137 static void btnTaskGest( void );
138 static void sys_shutdown( void );
139 static void startLogging (void);
140 // *****************************************************************************
141 // *****************************************************************************
142 // Section: Application Initialization and State Machine Functions
143 // *****************************************************************************
144 // *****************************************************************************
145
146 /*******************************************************************************
```

```
147     Function:
148       void APP_Initialize ( void )
149
150     Remarks:
151       See prototype in app.h.
152    */
153
154   void APP_Initialize ( void )
155   {
156       /* Keep the device ON */
157       PWR_HOLDOn();
158       LED_GOn();
159
160       // Start GNSS
161       //char gnssMessage[4+U_UBX_PROTOCOL_OVERHEAD_LENGTH_BYTES];
162       //char msgBody[4] = {0xFF, 0xFF, 0X09, 0x00};
163
164       // GNSS initialsiation data
165       /*char gnssMessage2[4+U_UBX_PROTOCOL_OVERHEAD_LENGTH_BYTES];
166       char msgBody2[13] = {0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00,
          0x00, 0x00, 0x01};
167       char msgBody3[4] = {0xFF, 0xFF, 0X09, 0x00};*/
168
169
170       // Initialization of the USART FIFOs
171       initFifo(&usartFifoRx, FIFO_RX_SIZE, a_fifoRx, 0);
172       initFifo(&usartFifoTx, FIFO_TX_SIZE, a_fifoTx, 0);
173
174       /* Start timers*/
175       DRV_TMR0_Start();
176       /* Init i2c bus */
177       i2c_init(1);
178
179       /* Reset GNSS*/
180       RESET_NOff();
181       BNO055_delay_msek(100);
182       /* Unreset GNSS */
183       RESET_NOn();
184       BNO055_delay_msek(300);
185
186       // Start GNSS
187       //uUbxProtocolEncode(0x06, 0x04, msgBody, 4, gnssMessage);
188       //serTransmitbuffer(USART_ID_2, gnssMessage, sizeof(gnssMessage));
189
190       /* Reset IMU */
191       RST_IMUOff();
192       BNO055_delay_msek(100);
193       RST_IMUOn();
194       BNO055_delay_msek(100);
195
196       // Reset interrupt pin
197       bno055_set_intr_rst(1);
198
199       /* Place the App state machine in its initial state. */
200       appData.state = APP_STATE_INIT;
201
202   }
203
204
205   /*******************************************************************************
206     Function:
207       void APP_Tasks ( void )
208
209     Remarks:
210       See prototype in app.h.
211    */
212
213   void APP_Tasks ( void )
214   {
215       /* Local bno055 data */
216       s_bno055_data bno055_local_data;
217       //s_gnssData gnss_ubx_local_data;
218       minmea_messages gnss_nmea_local_data;
```

```
219         //enum minmea_sentence_id gnss_nmea_msgId = MINMEA_UNKNOWN;
220         /* CONFIGURATION */
221         static char charRead[CHAR_READ_BUFFER_SIZE] = {0};
222         static uint32_t readCnt = 0;
223         static unsigned long oldIntG = 0;
224         static unsigned long oldIntI = 0;
225         static uint32_t oldInaPer = 0;
226         static bool oldLed = 0;
227         static int ledStateTemp = 0;
228 
229         // Character to send trough USART
230         static char charToSend = 0;
231 
232         /* Check the application's current state. */
233         switch ( appData.state )
234         {
235             /* Application's initial state. */
236             case APP_STATE_INIT:
237             {
238                 // Init delay
239                 BNO055_delay_msek(500);
240                 // Init and Measure set
241                 bno055_init_readout();
242                 BNO055_delay_msek(10);
243 
244 
245                 /* BNO055 motion interrupt mode */
246                 bno055_set_accel_any_motion_no_motion_axis_enable(
                    BNO055_ACCEL_ANY_MOTION_NO_MOTION_X_AXIS, BNO055_BIT_ENABLE);
247                 bno055_set_accel_any_motion_no_motion_axis_enable(
                    BNO055_ACCEL_ANY_MOTION_NO_MOTION_Y_AXIS, BNO055_BIT_ENABLE);
248                 bno055_set_accel_any_motion_no_motion_axis_enable(
                    BNO055_ACCEL_ANY_MOTION_NO_MOTION_Z_AXIS, BNO055_BIT_ENABLE);
249 
250                 bno055_set_accel_any_motion_durn(1);
251                 bno055_set_accel_any_motion_thres(25);
252 
253                 bno055_set_intr_accel_any_motion(BNO055_BIT_ENABLE);
254                 bno055_set_intr_mask_accel_any_motion(BNO055_BIT_ENABLE);
255                 bno055_set_intr_accel_no_motion(BNO055_BIT_DISABLE);
256 
257                 /*bno055_set_accel_slow_no_motion_enable(0);
258                 bno055_set_intr_accel_no_motion(BNO055_BIT_DISABLE);
259                 bno055_set_intr_mask_accel_no_motion(BNO055_BIT_DISABLE);*/
260 
261                 /* go to service task */
262                 appData.state = APP_STATE_CONFIG;
263                 /* Init ltime_BNO055 counter */
264                 timeData.ltime[BNO055_idx] = 0;
265                 break;
266             }
267             case APP_STATE_CONFIG:
268             {
269                 // Reset interrupt pin
270                 bno055_set_intr_rst(1);
271                 /* Init sd card parameters and read/create config File */
272                 sd_fat_cfg_init(&timeData.measPeriod[GNSS_idx], &timeData.measPeriod[
                    BNO055_idx], &appData.ledState, &timeData.inactivePeriod);
273 
274                 LED_GOff();
275 
276                 /* ---  Unmount timeout --- */
277                 if (ButtonMFStateGet())
278                     appData.state = APP_STATE_SHUTDOWN;
279                 break;
280             }
281             case APP_STATE_LOGGING:
282             {
283                 // BNO055 Measure routine
284                 if((timeData.measTodo[BNO055_idx] == true )&&(sd_logGetState() == APP_IDLE
                    ))
285                 {
286                     // If LED enabled
```

```
287                 if(appData.ledState == true){
288                     timeData.ledCnt = 0;
289                     LED_BOn();
290                 }
291                 /* BNO055 Read all important info routine */
292                 bno055_local_data.comres = bno055_read_routine(&bno055_local_data);
293                 /* Delta time */
294                 bno055_local_data.d_time = timeData.measCnt[BNO055_idx] - timeData.
                    ltime[BNO055_idx];
295                 /* Flag measure if acceleration detected */
296                 if((bno055_local_data.linear_accel.x >= 2*G) || (bno055_local_data.
                    linear_accel.y >= 2*G) || (bno055_local_data.linear_accel.z >= 2*G))
297                     bno055_local_data.flagImportantMeas = 1;
298                 else
299                     bno055_local_data.flagImportantMeas = 0;
300 
301                 /* Detect activity */
302                 if((bno055_local_data.linear_accel.x >= ACCEL_ACTIV_DETECT_msq)
303                    || (bno055_local_data.linear_accel.y >= ACCEL_ACTIV_DETECT_msq)
304                    || (bno055_local_data.linear_accel.z >= ACCEL_ACTIV_DETECT_msq))
305                     timeData.inactiveCnt = 0;
306 
307                 /* Write value to sdCard */
308                 sd_IMU_scheduleWrite(&bno055_local_data);
309                 /* Reset measure flag */
310                 timeData.measTodo[BNO055_idx] = false;
311                 /* Update last time counter */
312                 timeData.ltime[BNO055_idx] = timeData.measCnt[BNO055_idx];
313             }
314             // GNSS Measure routine
315             else if((timeData.measTodo[GNSS_idx] == true )&&(sd_logGetState() ==
                    APP_IDLE))
316             {
317                 /* Read GNSS position measure */
318                 //gnss_posGet_nmea(&gnss_nmea_local_data, &gnss_nmea_msgId);
319                 /* Write value to sdCard */
320                 sd_GNSS_scheduleWrite(&gnss_nmea_local_data);
321                 /* Reset measure flag */
322                 timeData.measTodo[GNSS_idx] = false;
323                             }
324             else
325             {
326                 /* No comm, so no error */
327                 bno055_local_data.comres = 0;
328                 //LED_BOff();
329             }
330 
331             /* If error detected : error LED */
332             if((bno055_local_data.comres != 0)||(sd_logGetState() == APP_MOUNT_DISK))
333                 LED_ROn();
334             else
335                 LED_ROff();
336 
337             /* --- SD FAT routine --- */
338             sd_fat_logging_task();
339             /* --- Button routine --- */
340             btnTaskGest();
341             /* --- Inactivity shutdown --- */
342             if (timeData.inactiveCnt >= (timeData.inactivePeriod*100))
343                 appData.state = APP_STATE_SHUTDOWN;
344 
345 
346             /* --- LIVE GNSS COMMAND --- */
347             if(pollSerialCmds(USART_ID_1, "glive", "GLIVE", "-lvg", "-LVG")){
348                 /* Stop SD card logging */
349                 stopLogging();
350                 /* USB communication states */
351                 appData.state = APP_STATE_COMM_LIVE_GNSS;
352                 LED_BOn();
353             }
354             /* --- LIVE IMU COMMAND --- */
355             if(pollSerialCmds(USART_ID_1, "ilive", "ILIVE", "-lvi", "-LVI")){
356                 /* Stop SD card logging */
```

```
357                 stopLogging();
358                 /* USB communication states */
359                 appData.state = APP_STATE_COMM_LIVE_IMU;
360                 LED_GOn();
361                 /* Deactivate USART2 (not used) */
362                 PLIB_USART_Disable(USART_ID_2);
363                 /* Reset measure flags and stop timer */
364                 DRV_TMR1_Start();
365             }
366 
367             /* --- SHUTDOWN SYSTEM COMMAND  --- */
368             if(pollSerialCmds(USART_ID_1, "shutdown", "SHUTDOWN", "-off", "-OFF")){
369 
370                 /* Turn off state */
371                 appData.state = APP_STATE_SHUTDOWN;
372             }
373 
374             /* --- CONFIG BLACKBOX --- */
375             if(pollSerialCmds(USART_ID_1, "config", "CONFIG", "-cfg", "-CFG")){
376                 // Stop SD card logging
377                 stopLogging();
378                 /* Deactivate USART2 (not used) */
379                 PLIB_USART_Disable(USART_ID_2);
380                 serTransmitString(USART_ID_1, "CONFIGURATION MODE \r\n");
381                 // Set config state to idle
382                 sd_cfgSetState(APP_CFG_IDLE);
383                 // Update configuration variables
384                 oldIntG = timeData.measPeriod[GNSS_idx];
385                 oldIntI = timeData.measPeriod[BNO055_idx];
386                 oldLed = appData.ledState;
387                 ledStateTemp = appData.ledState;
388                 // Turn off state
389                 appData.state = APP_STATE_CONFIGURATE_BBX;
390                 LED_GOn();
391             }
392 
393             /* --- GET GNSS LOGS --- */
394             if(pollSerialCmds(USART_ID_1, "glog", "GLOG", "-gl", "-GL")){
395                 // Display GNSS logs
396                 sd_fat_readDisplayFile("LOG_GNSS.txt");
397             }
398 
399             /* --- GEST IMU LOGS --- */
400             if(pollSerialCmds(USART_ID_1, "ilog", "ILOG", "-il", "-IL")){
401                 // Display IMU logs
402                 sd_fat_readDisplayFile("LOG_IMU.csv");
403             }
404 
405             /* --- DELETE COMMAND --- */
406             if(pollSerialCmds(USART_ID_1, "gclr", "GCLR", "-gc", "-GC")){
407                 // Delete file
408                 SYS_FS_FileDirectoryRemove("LOG_GNSS.txt");
409                 serTransmitString(USART_ID_1, "GNSS LOG DELETED \r\n");
410             }
411 
412             /* --- DELETE COMMAND --- */
413             if(pollSerialCmds(USART_ID_1, "iclr", "ICLR", "-ic", "-IC")){
414                 // Delete file
415                 SYS_FS_FileDirectoryRemove("LOG_IMU.csv");
416                 serTransmitString(USART_ID_1, "IMU LOG DELETED \r\n");
417             }
418 
419            break;
420         }
421         case APP_STATE_COMM_LIVE_GNSS:
422             /* No inactivity during this mode */
423             timeData.inactiveCnt = 0;
424             // Display GNSS live data trough USART 1
425             if(getReadSize(&usartFifoRx) > 0){
426                 getCharFromFifo(&usartFifoRx, &charToSend);
427                 PLIB_USART_TransmitterByteSend(USART_ID_1, charToSend);
428             }
             // If exit command detected, return to logging
```

```
429             if(pollSerialCmds(USART_ID_1, "exit", "EXIT", "x" ,"X"))
430                 startLogging();
431             break;
432         case APP_STATE_COMM_LIVE_IMU:
433             /* No inactivity during this mode */
434             timeData.inactiveCnt = 0;
435             // BNO055 Measure routine
436             if(timeData.measTodo[BNO055_idx] == true )
437             {
438                 // If LED enabled
439                 if(appData.ledState > 0){
440                     timeData.ledCnt = 0;
441                     LED_BOn();
442                 }
443                 /* BNO055 Read all important info routine */
444                 bno055_local_data.comres = bno055_read_routine(&bno055_local_data);
445                 /* Delta time */
446                 bno055_local_data.d_time = timeData.measCnt[BNO055_idx] - timeData.
                    ltime[BNO055_idx];
447
448                 /* Display readed values */
449                 serDisplayValues(&bno055_local_data);
450
451                 /* Reset measure flag */
452                 timeData.measTodo[BNO055_idx] = false;
453                 /* Update last time counter */
454                 timeData.ltime[BNO055_idx] = timeData.measCnt[BNO055_idx];
455             }
456             // If exit command detected, return to logging
457             if(pollSerialCmds(USART_ID_1, "exit", "EXIT", "x" ,"X")){
458                 startLogging();
459                 /* Reactivate USART2 (used) */
460                 PLIB_USART_Enable(USART_ID_2);
461             }
462
463             break;
464
465         case APP_STATE_CONFIGURATE_BBX:
466             /* No inactivity during this mode */
467             timeData.inactiveCnt = 0;
468             // Get command's characters
469             while(!(DRV_USART0_ReceiverBufferIsEmpty())&&(readCnt <
                CHAR_READ_BUFFER_SIZE)){
470                 charRead[readCnt] = PLIB_USART_ReceiverByteReceive(USART_ID_1);
471                 readCnt++;
472             }
473             // Command
474             if(readCnt >= CHAR_READ_BUFFER_SIZE)
475             {
476                 /* Reset read counter */
477                 readCnt = 0;
478                 /* Clear read buffer */
479                 memset(charRead,0,CHAR_READ_BUFFER_SIZE);
480             }
481
482             // Detect ENTER (End of command)
483             if(strstr(charRead, "\r") != NULL){
484                 // Scan command data
485                 sscanf(charRead, "INTG:%5lu", &timeData.measPeriod[GNSS_idx]);
486                 sscanf(charRead, "INTI:%5lu", &timeData.measPeriod[BNO055_idx]);
487                 sscanf(charRead, "LEDV:%2d", &ledStateTemp);
488                 sscanf(charRead, "TOFF:%5d", &timeData.inactivePeriod);
489                 // Cast int into boolean
490                 if (ledStateTemp > 0)
491                     appData.ledState = true;
492                 else
493                     appData.ledState = false;
494
495                 /* Reset read counter */
496                 readCnt = 0;
497                 /* Clear read buffer */
498                 memset(charRead,0,CHAR_READ_BUFFER_SIZE);
499             }
```

```
500                 // If config value changed
501                 if((timeData.measPeriod[GNSS_idx] != oldIntG) || (timeData.measPeriod[
                    BNO055_idx] != oldIntI) || (appData.ledState != oldLed)
502                     || (timeData.inactivePeriod != oldInaPer) ){
503
504                     serTransmitString(USART_ID_1, "COMMAND : VALUE CHANGED \r\n");
505                     // If data is not valid, keep the previous one
506                     if(timeData.measPeriod[GNSS_idx] <= 0){
507                         timeData.measPeriod[GNSS_idx] = oldIntG;
508                         serTransmitString(USART_ID_1, "ERROR GNSS VALUE <= 0 \r\n");
509                     }
510                     // If data is not valid, keep the previous one
511                     if(timeData.measPeriod[BNO055_idx] <= 0){
512                         timeData.measPeriod[BNO055_idx] = oldIntI;
513                         serTransmitString(USART_ID_1, "ERROR IMU VALUE <= 0 \r\n");
514                     }
515                     // If data is not valid, keep the previous one
516                     if(timeData.inactivePeriod <= 10){
517                         timeData.inactivePeriod = oldInaPer;
518                         serTransmitString(USART_ID_1, "ERROR INACTIVE PERIOD VALUE <= 10
                            \r\n");
519                     }
520                     /* Clear read buffer */
521                     memset(charRead,0,CHAR_READ_BUFFER_SIZE);
522                     // Write new config file
523                     sd_CFG_Write (timeData.measPeriod[GNSS_idx], timeData.measPeriod[
                        BNO055_idx], appData.ledState, timeData.inactivePeriod, true);
524                 }
525                 // Update polling config parameter
526                 oldIntG = timeData.measPeriod[GNSS_idx];
527                 oldIntI = timeData.measPeriod[BNO055_idx];
528                 oldLed = appData.ledState;
529                 oldInaPer = timeData.inactivePeriod;
530
531                 // Check occurence with commands
532                 if((strstr(charRead, "exit") != NULL)||(strstr(charRead, "EXIT") != NULL)
533                     || (strstr(charRead, "x") != NULL) || (strstr(charRead, "X") != NULL))
                         {
534                     /* Command detected */
535                     startLogging();
536                     /* Clear read buffer */
537                     memset(charRead,0,CHAR_READ_BUFFER_SIZE);
538                     /* Reset read counter */
539                     readCnt = 0;
540                     /* Reactivate USART2 (used) */
541                     PLIB_USART_Enable(USART_ID_2);
542                     break;
543                 }
544                 // Manipulate config file
545                 sd_fat_config_task ( false );
546                 break;
547
548             case APP_STATE_SHUTDOWN:
549             {
550                 /* Save and shutdown system */
551                 sys_shutdown();
552                 break;
553             }
554
555             /* The default state should never be executed. */
556             default:
557             {
558                 /* TODO: Handle error in application's state machine. */
559                 break;
560             }
561         }
562     }
563
564     void appStateSet( APP_STATES newState ){
565          appData.state = newState;
566     }
567
568     static void btnTaskGest( void ){
```

```
569         static bool Hold = false;
570         /* Button management : if rising edge detected */
571         if(((ButtonMFStateGet()))||(Hold == true))
572         {
573             /* Hold until falling edge */
574             Hold = true;
575             /* Start counting pressed time */
576             timeData.flagCntBtnPressed = true;
577             /* If falling edge detected */
578             if (ButtonMFStateGet() == 0)
579             {
580                 /* Reset flag and switchdescr */
581                 timeData.flagCntBtnPressed = false;
582                 DebounceClearReleased(&switchDescr);
583                 /* If pressed more time than power off */
584                 if(timeData.cntBtnPressed >= BTN_HOLD_SHUTDOWN_x10ms){
585                     /* Power off the system */
586                     appData.state = APP_STATE_SHUTDOWN;
587                 }
588                 timeData.cntBtnPressed = 0;
589                 Hold = false;
590             }
591         }
592     }
593
594     static void sys_shutdown( void ) {
595         /* Display shutting off mode */
596         LED_BOff();
597         LED_GOff();
598         LED_ROn();
599
600         /* If and SD card is mounted */
601         if(sd_logGetState() != APP_MOUNT_DISK){
602             /* Wait until SD availaible */
603             while(sd_logGetState() != APP_IDLE){
604                 /* SD FAT routine */
605                 sd_fat_logging_task();
606             }
607             /* Unmount disk */
608             sd_logSetState(APP_UNMOUNT_DISK);
609             /* Wait until unmounted*/
610             while(sd_logGetState() != APP_IDLE){
611                 sd_fat_logging_task();
612             }
613         }
614         /* Set acceleration only operation to save power */
615         bno055_set_operation_mode(BNO055_OPERATION_MODE_ACCONLY);
616         /* set the power mode as LOW POWER*/
617         bno055_set_power_mode(BNO055_POWER_MODE_LOWPOWER);
618         bno055_set_intr_accel_no_motion(BNO055_BIT_DISABLE);
619         // Reset interrupt pin
620         bno055_set_intr_rst(1);
621         do{
622             /* turn off the device */
623             PWR_HOLDOff();
624         }while(ButtonMFStateGet() == 0);
625     }
626
627     static void stopLogging (void)
628     {
629         /* Reset measure flags and stop timer */
630         DRV_TMR1_Stop();
631         timeData.measTodo[GNSS_idx] = false;
632         timeData.measTodo[BNO055_idx] = false;
633
634         /* Finish config */
635         while(sd_cfgGetState() != APP_CFG_IDLE){
636             sd_fat_cfg_init(&timeData.measPeriod[GNSS_idx], &timeData.measPeriod[
            BNO055_idx], &appData.ledState, &timeData.inactivePeriod);
637         }
638
639         /* Finish logging */
640         while(sd_logGetState() != APP_IDLE){
```

```
641             sd_fat_logging_task();
642         }
643 
644         /* Reset Leds states */
645         LED_ROff();
646         LED_ROff();
647         LED_GOff();
648     }
649 
650     static void startLogging (void)
651     {
652         // Logging state
653         appData.state = APP_STATE_LOGGING;
654         // Restart timer 1
655         DRV_TMR1_Start();
656         /* Reset Leds states */
657         LED_ROff();
658         LED_ROff();
659         LED_GOff();
660     }
661 
662     /*******************************************************************************
663      End of File
664      */
665 
```

```
/*******************************************************************************
  MPLAB Harmony Application Header File

  Company:
    Microchip Technology Inc.

  File Name:
    app.h

*******************************************************************************/

//DOM-IGNORE-BEGIN
/*******************************************************************************
Copyright (c) 2013-2014 released Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
 *******************************************************************************/
//DOM-IGNORE-END

#ifndef _APP_H
#define _APP_H

// *****************************************************************************
// *****************************************************************************
// Section: Included Files
// *****************************************************************************
// *****************************************************************************

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "system_config.h"
#include "system_definitions.h"
#include "bno055.h"

// DOM-IGNORE-BEGIN
#ifdef __cplusplus  // Provide C++ Compatibility

extern "C" {

#endif
// DOM-IGNORE-END


#define TIME_OUT                    80000000U
#define BTN_HOLD_SHUTDOWN_x10ms     200
#define NB_MEASURES                 2

#define ACCEL_ACTIV_DETECT_msq      0.3
#define T_CONFIG_TIMEOUT            20
#define T_INACTIVE_PERIOD_DEFAULT   20UL
#define T_INTERVAL_GNSS_DEFAULT     5000UL
#define T_INTERVAL_IMU_DEFAULT      500UL
#define LED_STATE_DEFAULT           (uint8_t)1

```

```
74  #define LED_PERIOD                  5
75
76  #define CHAR_READ_BUFFER_SIZE       30
77
78  #define G                 9.81
79
80
81  // *****************************************************************************
82  // *****************************************************************************
83  // Section: Type Definitions
84  // *****************************************************************************
85  // *****************************************************************************
86  typedef struct {
87      s32 comres;
88      bool flagMeasReady;
89      uint8_t flagImportantMeas;
90      struct bno055_gravity_double_t gravity;
91      struct bno055_linear_accel_double_t linear_accel;
92      struct bno055_euler_double_t euler;
93      struct bno055_gyro_double_t gyro;
94      struct bno055_mag_double_t mag;
95      struct bno055_quaternion_t quaternion;
96      unsigned long time;
97      unsigned long l_time;
98      uint16_t d_time;
99  }s_bno055_data;
100 // *****************************************************************************
101 /* Application states
102
103   Summary:
104     Application states enumeration
105
106   Description:
107     This enumeration defines the valid application states.  These states
108     determine the behavior of the application at various times.
109 */
110
111 typedef enum
112 {
113     /* Application's state machine's initial state. */
114     APP_STATE_INIT=0,
115     APP_STATE_CONFIG,
116     APP_STATE_LOGGING,
117     APP_STATE_FLAG_MEAS,
118     APP_STATE_COMM_LIVE_GNSS,
119     APP_STATE_COMM_LIVE_IMU,
120     APP_STATE_CONFIGURATE_BBX,
121     APP_STATE_SHUTDOWN
122     /* TODO: Define states used by the application state machine. */
123
124 } APP_STATES;
125
126
127 // *****************************************************************************
128 /* Application Data
129
130   Summary:
131     Holds application data
132
133   Description:
134     This structure holds the application's data.
135
136   Remarks:
137     Application strings and buffers are be defined outside this structure.
138  */
139
140 typedef struct
141 {
142     /* The application's current state */
143     APP_STATES state;
144
145     bool ledState;
146
```

```
147         /* TODO: Define any additional data used by the application. */
148
149     } APP_DATA;
150
151     typedef struct
152     {
153         /* DELAY DATA */
154         bool tmrTickFlag;
155         unsigned long delayCnt;
156
157         /* MEASURES DATA */
158         unsigned long measCnt[NB_MEASURES];
159         unsigned long ltime[NB_MEASURES];
160         bool measTodo[NB_MEASURES];
161         unsigned long measPeriod[NB_MEASURES];
162
163         unsigned long inactiveCnt;
164         uint32_t inactivePeriod;
165
166         /* DISPLAY DATA */
167         uint32_t ledCnt;
168
169         /* BUTTON DATA */
170         bool flagCntBtnPressed;
171         uint32_t cntBtnPressed;
172     }TIMER_DATA;
173
174     /* Measures index */
175     enum measure{BNO055_idx, GNSS_idx};
176
177     // *****************************************************************************
178     // *****************************************************************************
179     // Section: Application Callback Routines
180     // *****************************************************************************
181     // *****************************************************************************
182     /* These routines are called by drivers when certain events occur.
183     */
184
185     // *****************************************************************************
186     // *****************************************************************************
187     // Section: Application Initialization and State Machine Functions
188     // *****************************************************************************
189     // *****************************************************************************
190
191     /*******************************************************************************
192       Function:
193         void APP_Initialize ( void )
194
195       Summary:
196          MPLAB Harmony application initialization routine.
197
198       Description:
199         This function initializes the Harmony application.  It places the
200         application in its initial state and prepares it to run so that its
201         APP_Tasks function can be called.
202
203       Precondition:
204         All other system initialization routines should be called before calling
205         this routine (in "SYS_Initialize").
206
207       Parameters:
208         None.
209
210       Returns:
211         None.
212
213       Example:
214         <code>
215         APP_Initialize();
216         </code>
217
218       Remarks:
219         This routine must be called from the SYS_Initialize function.
```

```
220   */
221
222   void APP_Initialize ( void );
223
224
225   /*******************************************************************************
226     Function:
227       void APP_Tasks ( void )
228
229     Summary:
230       MPLAB Harmony Demo application tasks function
231
232     Description:
233       This routine is the Harmony Demo application's tasks function.  It
234       defines the application's state machine and core logic.
235
236     Precondition:
237       The system and application initialization ("SYS_Initialize") should be
238       called before calling this.
239
240     Parameters:
241       None.
242
243     Returns:
244       None.
245
246     Example:
247       <code>
248       APP_Tasks();
249       </code>
250
251     Remarks:
252       This routine must be called from SYS_Tasks() routine.
253    */
254
255   void APP_Tasks( void );
256
257   // CALLBACKS
258   void delayTimer_callback( void );
259   void stateTimer_callback( void );
260
261   void appStateSet( APP_STATES newState );
262
263   #endif /* _APP_H */
264
265   //DOM-IGNORE-BEGIN
266   #ifdef __cplusplus
267   }
268   #endif
269   //DOM-IGNORE-END
270
271   /*******************************************************************************
272    End of File
273    */
274
275
```

```
/*******************************************************************************
 System Interrupts File

  File Name:
    system_interrupt.c

  Summary:
    Raw ISR definitions.

  Description:
    This file contains a definitions of the raw ISRs required to support the
    interrupt sub-system.

  Summary:
    This file contains source code for the interrupt vector functions in the
    system.

  Description:
    This file contains source code for the interrupt vector functions in the
    system.  It implements the system and part specific vector "stub" functions
    from which the individual "Tasks" functions are called for any modules
    executing interrupt-driven in the MPLAB Harmony system.

  Remarks:
    This file requires access to the systemObjects global data structure that
    contains the object handles to all MPLAB Harmony module objects executing
    interrupt-driven in the system.  These handles are passed into the individual
    module "Tasks" functions to identify the instance of the module to maintain.
 *******************************************************************************/

// DOM-IGNORE-BEGIN
/*******************************************************************************
Copyright (c) 2011-2014 released Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
 *******************************************************************************/
// DOM-IGNORE-END

// *****************************************************************************
// *****************************************************************************
// Section: Included Files
// *****************************************************************************
// *****************************************************************************

#include "system/common/sys_common.h"
#include "app.h"
#include "system_definitions.h"
#include "usart_FIFO.h"

// *****************************************************************************
// *****************************************************************************
// Section: System Interrupt Vector Functions
// *****************************************************************************
// *****************************************************************************
void __ISR(_UART_1_VECTOR, ipl0AUTO) _IntHandlerDrvUsartInstance0(void)
{
```

```
74      DRV_USART_TasksTransmit(sysObj.drvUsart0);
75      DRV_USART_TasksError(sysObj.drvUsart0);
76      DRV_USART_TasksReceive(sysObj.drvUsart0);
77  }
78
79
80
81
82  void __ISR(_UART_2_VECTOR, ipl1AUTO) _IntHandlerDrvUsartInstance1(void)
83  {
84      USART_ERROR usartStatus;
85      bool        isTxBuffFull;
86      char      charReceived;
87      char      charToSend;
88      char      TXsize;
89
90       //------------------------------------------------------------------------// RX
         interrupt
91      if(PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_USART_2_RECEIVE) &&
92                  PLIB_INT_SourceIsEnabled(INT_ID_0, INT_SOURCE_USART_2_RECEIVE)){
93
94          // Parity error or overrun
95          usartStatus = PLIB_USART_ErrorsGet(USART_ID_2);
96
97          if ((usartStatus & (USART_ERROR_PARITY | USART_ERROR_FRAMING |
98                  USART_ERROR_RECEIVER_OVERRUN)) == 0){
99
100             // All char received are transferred to the FIFO
101             // 1 if ONE_CHAR, 4 if HALF_FULL and 6 3B4FULL
102             while(PLIB_USART_ReceiverDataIsAvailable(USART_ID_2)){
103
104                 charReceived = PLIB_USART_ReceiverByteReceive(USART_ID_2);
105                 putCharInFifo(&usartFifoRx, charReceived);
106             }
107             // Buffer is empty, clear interrupt flag
108             PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_2_RECEIVE);
109
110         }else{
111             // Deleting errors
112             // Reading errors clears them except for overrun
113             if((usartStatus & USART_ERROR_RECEIVER_OVERRUN) ==
114                     USART_ERROR_RECEIVER_OVERRUN){
115
116                 PLIB_USART_ReceiverOverrunErrorClear(USART_ID_2);
117             }
118         }
119     }
120
121
122     //------------------------------------------------------------------------// TX
        interrupt
123     if (PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_USART_2_TRANSMIT) &&
124                 PLIB_INT_SourceIsEnabled(INT_ID_0, INT_SOURCE_USART_2_TRANSMIT)){
125
126         TXsize = getReadSize(&usartFifoTx);
127         // i_cts = input(RS232_CTS);
128
129         isTxBuffFull = PLIB_USART_TransmitterBufferIsFull(USART_ID_2);
130
131         if (/*(i_cts == 0) && */(TXsize > 0) && (isTxBuffFull == false)){
132             do{
133                 getCharFromFifo(&usartFifoTx, &charToSend);
134                 if(charToSend != '\0') PLIB_USART_TransmitterByteSend(USART_ID_2,
                    charToSend);
135                 /*i_cts = RS232_CTS;*/
136                 TXsize = getReadSize (&usartFifoTx);
137                 isTxBuffFull = PLIB_USART_TransmitterBufferIsFull(USART_ID_2);
138             }while(/*(i_cts == 0) && */( TXsize > 0 ) && isTxBuffFull == false);
139         }
140
141         // Disables TX interrupt (to avoid unnecessary interruptions if there's
142         // nothing left to transmit)
143         if(TXsize == 0){
```

```
144
145                 PLIB_INT_SourceDisable(INT_ID_0, INT_SOURCE_USART_2_TRANSMIT);
146             }
147             // Clears the TX interrupt Flag
148             PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_2_TRANSMIT);
149         }
150     }
151
152     void __ISR(_TIMER_1_VECTOR, ipl6AUTO) IntHandlerDrvTmrInstance0(void)
153     {
154         PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_1);
155         delayTimer_callback();
156     }
157     void __ISR(_TIMER_2_VECTOR, ipl5AUTO) IntHandlerDrvTmrInstance1(void)
158     {
159         PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_2);
160         stateTimer_callback();
161     }
162
163     void __ISR(_SPI_1_VECTOR, ipl1AUTO) _IntHandlerSPIInstance0(void)
164     {
165         DRV_SPI_Tasks(sysObj.spiObjectIdx0);
166     }
167     /*******************************************************************************
168      End of File
169     */
170
```

```c
/* ************************************************************************ */
/** Descriptive File Name

  @Company
    ETML-ES

  @File Name
    sd_fat_gest.c

  @Summary
    SD card fat system management

  @Description
    SD card fat system management
 */
/* ************************************************************************ */

/* ************************************************************************ */
/* ************************************************************************ */
/* Section: Included Files                                                  */
/* ************************************************************************ */
/* ************************************************************************ */

/* This section lists the other files that are included in this file.
 */

#include "Mc32_sdFatGest.h"
#include <stdio.h>
#include "app.h"
#include "bno055_support.h"
#include "GNSS/u_gnss_pos.h"
#include <stdio.h>
#include "usart_FIFO.h"
#include "MC32_serComm.h"

/* ************************************************************************ */
/* ************************************************************************ */
/* Section: File Scope or Global Data                                       */
/* ************************************************************************ */
/* ************************************************************************ */

APP_FAT_DATA COHERENT_ALIGNED appFatData;
/* ************************************************************************ */
/** Descriptive Data Item Name

  @Summary
    Brief one-line summary of the data item.

  @Description
    Full description, explaining the purpose and usage of data item.
    <p>
    Additional description in consecutive paragraphs separated by HTML
    paragraph breaks, as necessary.
    <p>
    Type "JavaDoc" in the "How Do I?" IDE toolbar for more information on tags.

  @Remarks
    Any additional remarks
 */


/* ************************************************************************ */
/* ************************************************************************ */
// Section: Local Functions                                                 */
/* ************************************************************************ */
/* ************************************************************************ */
// Function prototype
static uint8_t parseConfig(unsigned long *tGnss, unsigned long *tImu, bool *ledState,
unsigned long *tInactive);
/* ************************************************************************ */

/* ************************************************************************ */
/* ************************************************************************ */
```

```
73  // Section: Interface Functions                                              */
74  /* ************************************************************************** */
75  /* ************************************************************************** */
76
77  void sd_fat_config_task ( bool init )
78  {
79      /* The application task cfg_state machine */
80      switch(appFatData.cfg_state)
81      {
82          case APP_CFG_MOUNT_DISK:
83              if(SYS_FS_Mount("/dev/mmcblka1", "/mnt/myDrive", FAT, 0, NULL) != 0)
84              {
85                  /* The disk could not be mounted. Try
86                   * mounting again untill success. */
87                  LED_ROn();
88                  appFatData.cfg_state = APP_CFG_MOUNT_DISK;
89              }
90              else
91              {
92                  /* Mount was successful. Unmount the disk, for testing. */
93                  LED_ROff();
94                  appFatData.cfg_state = APP_CFG_SET_CURRENT_DRIVE;
95              }
96              break;
97
98          case APP_CFG_SET_CURRENT_DRIVE:
99              if(SYS_FS_CurrentDriveSet("/mnt/myDrive") == SYS_FS_RES_FAILURE)
100             {
101                 /* Error while setting current drive */
102                 appFatData.cfg_state = APP_CFG_ERROR;
103             }
104             else
105             {
106                 if(init == true)
107                     /* Open config file for reading. */
108                     appFatData.cfg_state = APP_CFG_OPEN_READ_CONFIG_FILE;
109                 else
110                     /* Wait for further commands. */
111                     appFatData.cfg_state = APP_CFG_IDLE;
112             }
113             break;
114
115         case APP_CFG_OPEN_READ_CONFIG_FILE:
116             appFatData.fileCfgHandle = SYS_FS_FileOpen("CONFIG.txt",
117                     (SYS_FS_FILE_OPEN_READ));
118             if(appFatData.fileCfgHandle == SYS_FS_HANDLE_INVALID)
119             {
120                 /* No config file, write default config file */
121                 sd_CFG_Write(T_INTERVAL_GNSS_DEFAULT, T_INTERVAL_IMU_DEFAULT,
                    LED_STATE_DEFAULT, T_INACTIVE_PERIOD_DEFAULT, true);
122
123                 /* Re-try to open file as read */
124                 //appFatData.cfg_state = APP_CFG_OPEN_READ_CONFIG_FILE;
125
126             }
127             else
128             {
129                 /* Create a directory. */
130                 appFatData.cfg_state = APP_CFG_READ_CONFIG_FILE;
131             }
132             break;
133
134         case APP_CFG_READ_CONFIG_FILE:
135             /* If read was success, try writing to the new file */
136             if(SYS_FS_FileRead(appFatData.fileCfgHandle, appFatData.cfg_data,
137                     SYS_FS_FileSize(appFatData.fileCfgHandle)) == -1)
138             {
139                 /* Write was not successful. Close the file
140                  * and error out.*/
141                 SYS_FS_FileClose(appFatData.fileCfgHandle);
142                 appFatData.cfg_state = APP_CFG_ERROR;
143             }
144             else
```

```
145             {
146                 appFatData.cfg_state = APP_CFG_CLOSE_FILE;
147             }
148             break;
149         case APP_CFG_OPEN_WRITE_CONFIG_FILE:
150             appFatData.fileCfgHandle = SYS_FS_FileOpen("CONFIG.txt",
151                     (SYS_FS_FILE_OPEN_WRITE));
152             if(appFatData.fileCfgHandle == SYS_FS_HANDLE_INVALID)
153             {
154                 /* Could not open the file. Error out*/
155                 appFatData.cfg_state = APP_CFG_ERROR;
156             }
157             else
158             {
159                 /* Create a directory. */
160                 appFatData.cfg_state = APP_CFG_WRITE_CONFIG_FILE;
161             }
162             break;
163 
164         case APP_CFG_WRITE_CONFIG_FILE:
165             /* If read was success, try writing to the new file */
166             if(SYS_FS_FileStringPut(appFatData.fileCfgHandle, appFatData.cfg_data)== -
                1)
167             {
168                 /* Write was not successful. Close the file
169                  * and error out.*/
170                 SYS_FS_FileClose(appFatData.fileCfgHandle);
171                 appFatData.cfg_state = APP_CFG_ERROR;
172             }
173             else
174             {
175                 appFatData.cfg_state = APP_CFG_CLOSE_FILE;
176             }
177             break;
178         case APP_CFG_CLOSE_FILE:
179             /* Close the file */
180             SYS_FS_FileClose(appFatData.fileCfgHandle);
181              /* The test was successful. Lets idle. */
182             if(init == true)
183                 appFatData.cfg_state = APP_CFG_UNMOUNT_DISK;
184             else
185                 appFatData.cfg_state = APP_CFG_IDLE;
186             break;
187 
188         case APP_CFG_IDLE:
189             /* The appliction comes here when the demo
190              * has completed successfully. Switch on
191              * green LED. */
192             //BSP_LEDOn(APP_SUCCESS_LED);
193             LED_ROff();
194             break;
195         case APP_CFG_ERROR:
196             /* The appliction comes here when the demo
197              * has failed. Switch on the red LED.*/
198             //BSP_LEDOn(APP_FAILURE_LED);
199             LED_ROn();
200             break;
201         default:
202             break;
203 
204         case APP_CFG_UNMOUNT_DISK:
205             if(SYS_FS_Unmount("/mnt/myDrive") != 0)
206             {
207                 /* The disk could not be un mounted. Try
208                  * un mounting again untill success. */
209 
210                 appFatData.cfg_state = APP_CFG_UNMOUNT_DISK;
211             }
212             else
213             {
214                 /* UnMount was successful. Mount the disk again */
215                 appFatData.cfg_state = APP_CFG_IDLE;
216             }
```

```
217                 break;
218         }
219     }
220
221     // Loggin task
222     void sd_fat_logging_task ( void )
223     {
224         /* The application task log_state machine */
225         switch(appFatData.log_state)
226         {
227             case APP_MOUNT_DISK:
228                 if(SYS_FS_Mount("/dev/mmcblka1", "/mnt/myDrive", FAT, 0, NULL) != 0)
229                 {
230                     /* The disk could not be mounted. Try
231                      * mounting again untill success. */
232
233                     appFatData.log_state = APP_MOUNT_DISK;
234                 }
235                 else
236                 {
237                     /* Mount was successful. Unmount the disk, for testing. */
238
239                     appFatData.log_state = APP_SET_CURRENT_DRIVE;
240                 }
241                 break;
242
243             case APP_SET_CURRENT_DRIVE:
244                 if(SYS_FS_CurrentDriveSet("/mnt/myDrive") == SYS_FS_RES_FAILURE)
245                 {
246                     /* Error while setting current drive */
247                     appFatData.log_state = APP_ERROR;
248                 }
249                 else
250                 {
251                     /* Open a file for reading. */
252                     appFatData.log_state = APP_IDLE;
253                 }
254                 break;
255
256             case APP_WRITE_MEASURE_FILE:
257                 appFatData.fileMeasureHandle = SYS_FS_FileOpen(appFatData.fileName,
258                         (SYS_FS_FILE_OPEN_APPEND_PLUS));
259                 if(appFatData.fileMeasureHandle == SYS_FS_HANDLE_INVALID)
260                 {
261                     /* Could not open the file. Error out*/
262                     appFatData.log_state = APP_ERROR;
263                 }
264                 else
265                 {
266                     /* Create a directory. */
267                     appFatData.log_state = APP_WRITE_TO_MEASURE_FILE;
268                 }
269                 break;
270
271             case APP_WRITE_TO_MEASURE_FILE:
272                 /* If read was success, try writing to the new file */
273                 if(SYS_FS_FileStringPut(appFatData.fileMeasureHandle, appFatData.data) ==
                    -1)
274                 {
275                     /* Write was not successful. Close the file
276                      * and error out.*/
277                     SYS_FS_FileClose(appFatData.fileMeasureHandle);
278                     appFatData.log_state = APP_ERROR;
279                 }
280                 else
281                 {
282                     appFatData.log_state = APP_CLOSE_FILE;
283                 }
284                 break;
285
286             case APP_CLOSE_FILE:
287                 /* Close both files */
288                 SYS_FS_FileClose(appFatData.fileMeasureHandle);
```

```
289                 /* The test was successful. Lets idle. */
290                 appFatData.log_state = APP_IDLE;
291                 break;
292 
293             case APP_IDLE:
294                 /* The appliction comes here when the demo
295                  * has completed successfully. Switch on
296                  * green LED. */
297                 //BSP_LEDOn(APP_SUCCESS_LED);
298                 LED_ROff();
299                 break;
300             case APP_ERROR:
301                 /* The appliction comes here when the demo
302                  * has failed. Switch on the red LED.*/
303                 //BSP_LEDOn(APP_FAILURE_LED);
304                 LED_ROn();
305                 break;
306             default:
307                 break;
308 
309             case APP_UNMOUNT_DISK:
310                 if(SYS_FS_Unmount("/mnt/myDrive") != 0)
311                 {
312                     /* The disk could not be un mounted. Try
313                      * un mounting again untill success. */
314 
315                     appFatData.log_state = APP_UNMOUNT_DISK;
316                 }
317                 else
318                 {
319                     /* UnMount was successful. Mount the disk again */
320                     appFatData.log_state = APP_IDLE;
321                 }
322                 break;
323 
324         }
325 
326     //    SYS_FS_Tasks();
327     } //End of APP_Tasks
328 
329     void sd_IMU_scheduleWrite (s_bno055_data * data)
330     {
331         /* If sd Card available */
332         if(appFatData.log_state == APP_IDLE)
333         {
334             // Prepare file name
335             sprintf(appFatData.fileName, "LOG_IMU.csv");
336             /* Next log_state : write to file */
337             appFatData.log_state = APP_WRITE_MEASURE_FILE;
338             /* Write the buffer */
339             sprintf(appFatData.data,
                    "%d;%d0;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;
                    %.4f;%d;%d;%d;%d;\r\n"
340                                     ,data->flagImportantMeas, (data->d_time), data->
                                        gravity.x, data->gravity.y, data->gravity.z, data->
                                        gyro.x, data->gyro.y, data->gyro.z
341                                     ,data->mag.x, data->mag.y, data->mag.z, data->
                                        linear_accel.x, data->linear_accel.y, data->
                                        linear_accel.z
342                                     ,data->euler.h, data->euler.p, data->euler.r, data->
                                        quaternion.w, data->quaternion.x, data->quaternion.y
                                        , data->quaternion.z);
343             /* Compute the number of bytes to send */
344             appFatData.nBytesToWrite = strlen(appFatData.data);
345         }
346     }
347 
348     void sd_GNSS_scheduleWrite (minmea_messages * pGnssData)
349     {
350         char fifoBuffer[FIFO_RX_SIZE];
351         /* If sd Card available */
352         if(appFatData.log_state == APP_IDLE)
353         {
```

```
354             // Prepare file name
355             sprintf(appFatData.fileName, "LOG_GNSS.txt");
356             /* Next log_state : write to file */
357             appFatData.log_state = APP_WRITE_MEASURE_FILE;
358             /* Write the buffer */
359             getFifoToLastReturn(&usartFifoRx, fifoBuffer);
360 
361             sprintf(appFatData.data, "%s", fifoBuffer);
362 
363             //sprintf(appFatData.data, "%s",  );
364             /* Compute the number of bytes to send */
365             appFatData.nBytesToWrite = strlen(appFatData.data);
366         }
367     }
368 
369     void sd_CFG_Write (uint32_t tLogGNSS_ms, uint32_t tLogIMU_ms, uint8_t ledState,
    uint32_t tInactiveP, bool skipMount)
370     {
371         /* If sd Card available */
372         if((appFatData.cfg_state == APP_CFG_IDLE)||(appFatData.cfg_state ==
        APP_CFG_OPEN_READ_CONFIG_FILE))
373         {
374             /* Close the file */
375             SYS_FS_FileClose(appFatData.fileCfgHandle);
376 
377             if(skipMount == false)
378                 /* Next config : mount disk */
379                 appFatData.cfg_state = APP_CFG_MOUNT_DISK;
380             else if(skipMount == true)
381                 /* Next config : write to file */
382                 appFatData.cfg_state = APP_CFG_OPEN_WRITE_CONFIG_FILE;
383 
384             /* Write the buffer */
385             sprintf(appFatData.cfg_data, "$LOG INTERVAL GNSS [ms] : %u\r\n$LOG INTERVAL
            IMU [ms] : %u\r\n$LED ENABLE [1/0] : %u\r\n$INACTIVE PERIOD [s] : %u\r\n",
386                     tLogGNSS_ms, tLogIMU_ms, ledState, tInactiveP);
387             /* Compute the number of bytes to send */
388             appFatData.nBytesToWrite = strlen(appFatData.cfg_data);
389 
390         }
391     }
392 
393 
394     APP_FAT_LOG_STATES sd_logGetState( void )
395     {
396         return appFatData.log_state;
397     }
398 
399     void sd_logSetState( APP_FAT_LOG_STATES newState )
400     {
401         appFatData.log_state = newState;
402     }
403 
404     // CONFIG FUNCTIONS
405 
406     APP_FAT_CONFIG_STATES sd_cfgGetState( void )
407     {
408         return appFatData.cfg_state;
409     }
410     void sd_cfgSetState( APP_FAT_CONFIG_STATES newState )
411     {
412          appFatData.cfg_state = newState;
413     }
414 
415     char* sd_cfgGetCfgBuffer( void )
416     {
417         return appFatData.cfg_data;
418     }
419 
420     void sd_fat_cfg_init(unsigned long *tGnss, unsigned long *tImu, bool *ledState,
    uint32_t *tInactivePeriod)
421     {
422         // Config parser error
```

```
423         uint8_t parseError = 0;
424         unsigned long tGnssLocal = 0;
425         unsigned long tImuLocal = 0;
426         unsigned long tInactive = 0;
427         bool ledStateLocal = 0;
428 
429         //appFatData.nBytesRead = 0;
430         //appFatData.nBytesToWrite = 0;
431 
432         //appFatData.log_state = APP_MOUNT_DISK;
433         //appFatData.cfg_state = APP_CFG_MOUNT_DISK;
434 
435         // Read config routine, until error or success
436         sd_fat_config_task(true);
437 
438         // If read config routine was a success
439         if(sd_cfgGetState() == APP_CFG_IDLE)
440             // Parse config buffer to get parameters
441             parseError = parseConfig(&tGnssLocal, &tImuLocal, &ledStateLocal, &tInactive);
442         // If the parsing failed or the read config routine failed
443         if((parseError > 0)||(sd_cfgGetState() == APP_CFG_ERROR))
444         {
445             // Set default system parameters
446             *tGnss = T_INTERVAL_GNSS_DEFAULT;
447             *tImu = T_INTERVAL_IMU_DEFAULT;
448             *ledState = LED_STATE_DEFAULT;
449             *tInactivePeriod = T_INACTIVE_PERIOD_DEFAULT;
450             appStateSet(APP_STATE_LOGGING);
451             // Start measure timer
452             DRV_TMR1_Start();
453         }
454         else if ((sd_cfgGetState() == APP_CFG_IDLE))
455         {
456             *tGnss = tGnssLocal;
457             *tImu = tImuLocal;
458             *ledState = ledStateLocal;
459             *tInactivePeriod = tInactive;
460             appStateSet(APP_STATE_LOGGING);
461             // Start measure timer
462             DRV_TMR1_Start();
463         }
464     }
465 
466     static uint8_t parseConfig(unsigned long *tGnss, unsigned long *tImu, bool *ledState,
    unsigned long *tInactive)
467     {
468         char *ptBufferHead;
469         char *ptBufferTail;
470         char ptTrame[10];
471         uint8_t error = 0;
472 
473         // Locate the head and tail of the first data
474         ptBufferHead = strstr(appFatData.cfg_data, " :");
475         ptBufferTail = strstr(appFatData.cfg_data, "\r\n");
476         // Check if the pointers are corrects
477         if((ptBufferHead != NULL)&&(ptBufferTail != NULL)&&(ptBufferHead < ptBufferTail)){
478             // Copy the data between the head and the tail in a sub-pointer
479             strncpy(ptTrame, (ptBufferHead+2), (ptBufferTail-ptBufferHead));
480             // Convert the character to value
481             *tGnss = (uint32_t) atoi(ptTrame);
482         }
483         else
484             error++;
485 
486         // Locate the head and tail of the first data
487         ptBufferHead = strstr(ptBufferTail, " :");
488         ptBufferTail = strstr(ptBufferHead, "\r\n");
489         // Check if the pointers are corrects
490         if((ptBufferHead != NULL)&&(ptBufferTail != NULL)&&(ptBufferHead < ptBufferTail)){
491             // Copy the data between the head and the tail in a sub-pointer
492             strncpy(ptTrame, (ptBufferHead+2), (ptBufferTail-ptBufferHead));
493             // Convert the character to value
494             *tImu = (uint32_t) atoi(ptTrame);
```

```
    }
    else
        error++;

    // Locate the head and tail of the first data
    ptBufferHead = strstr(ptBufferTail, " :");
    ptBufferTail = strstr(ptBufferHead, "\r\n");
    // Check if the pointers are corrects
    if((ptBufferHead != NULL)&&(ptBufferTail != NULL)&&(ptBufferHead < ptBufferTail)){
        // Copy the data between the head and the tail in a sub-pointer
        strncpy(ptTrame, (ptBufferHead+2), (ptBufferTail-ptBufferHead));
        // Convert the character to value
        *ledState = (bool) atoi(ptTrame);
    }
    else
        error++;

    // Locate the head and tail of the first data
    ptBufferHead = strstr(ptBufferTail, " :");
    ptBufferTail = strstr(ptBufferHead, "\r\n");
    // Check if the pointers are corrects
    if((ptBufferHead != NULL)&&(ptBufferTail != NULL)&&(ptBufferHead < ptBufferTail)){
        // Copy the data between the head and the tail in a sub-pointer
        strncpy(ptTrame, (ptBufferHead+2), (ptBufferTail-ptBufferHead));
        // Convert the character to value
        *tInactive = (uint32_t) atoi(ptTrame);
    }
    else
        error++;

    return error;

}

void sd_fat_readDisplayFile(const char * fileName)
{
    const uint16_t READSIZE = 256;
    uint32_t i = 0;
    char stringRead[READSIZE];
    unsigned long cntTimeaout = 0;

    /* Close both files */
    SYS_FS_FileClose(appFatData.fileMeasureHandle);
    // Read config file
    appFatData.fileCfgHandle = SYS_FS_FileOpen(fileName,(SYS_FS_FILE_OPEN_READ));

    do{

        SYS_FS_FileStringGet(appFatData.fileCfgHandle, stringRead, READSIZE);

        do{
            if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))
            {
                PLIB_USART_TransmitterByteSend(USART_ID_1, stringRead[i]);
                i++;
            }
            cntTimeaout++;
        }while((i < strlen(stringRead))&&(cntTimeaout<TIME_OUT));

        i = 0;
        cntTimeaout = 0;

        if(pollSerialCmds(USART_ID_1, "exit", "EXIT", "x" ,"X"))
            break;

    }while(!SYS_FS_FileEOF(appFatData.fileCfgHandle));

    /* Close both files */
    SYS_FS_FileClose(appFatData.fileMeasureHandle);
}

//bool sd_fat_readFile(const char * fileName, char readBuffer[])
//{
```

```
568   //    uint32_t fileSize = 0;
569   //    static bool fullyRead = false;
570   //    /* Close both files */
571   //    SYS_FS_FileClose(appFatData.fileMeasureHandle);
572   //    // Read config file
573   //    appFatData.fileCfgHandle = SYS_FS_FileOpen(fileName,(SYS_FS_FILE_OPEN_READ));
574   //
575   //    fileSize = SYS_FS_FileSize(appFatData.fileCfgHandle);
576   //
577   //    if (fileSize <= sizeof(readBuffer))
578   //        SYS_FS_FileRead(appFatData.fileCfgHandle, readBuffer, fileSize);
579   //    else{
580   //
581   //    }
582   //    /* Close both files */
583   //    SYS_FS_FileClose(appFatData.fileMeasureHandle);
584   //}
585
586   /* *****************************************************************************
587    End of File
588    */
589
```

```
/*******************************************************************************
  MPLAB Harmony Application Header File

  Company:
    ETML-ES

  File Name:
    MC32_sdFatGest.h

*******************************************************************************/

//DOM-IGNORE-BEGIN

//DOM-IGNORE-END

#ifndef _SD_FAT_GEST_H
#define _SD_FAT_GEST_H


// *****************************************************************************
// *****************************************************************************
// Section: Included Files
// *****************************************************************************
// *****************************************************************************

#include "app.h"
#include "GNSS/minmea.h"
#include "usart_FIFO.h"
// *****************************************************************************
// *****************************************************************************
// Section: Type Definitions
// *****************************************************************************
// *****************************************************************************

#ifdef DRV_SDHC_USE_DMA
#define DATA_BUFFER_ALIGN            __attribute__((coherent, aligned(32)))
#else
#define DATA_BUFFER_ALIGN            __attribute__((aligned(32)))
#endif

// *****************************************************************************
/* Application States

  Summary:
    Application states enumeration

  Description:
    This enumeration defines the valid application states.  These states
    determine the behavior of the application at various times.
*/

typedef enum
{
    /* Application's state machine's initial state. */
    /* The app mounts the disk */
    APP_MOUNT_DISK = 0,

    /* Set the current drive */
    APP_SET_CURRENT_DRIVE,

    /* The app opens the file to read */
    APP_WRITE_MEASURE_FILE,

    /* The app reads from a file and writes to another file */
    APP_WRITE_TO_MEASURE_FILE,

    /* The app closes the file*/
    APP_CLOSE_FILE,

    /* The app closes the file and idles */
    APP_IDLE,

    /* An app error has occurred */
```

```
74          APP_ERROR,
75
76          /* Unmount disk */
77          APP_UNMOUNT_DISK
78
79      } APP_FAT_LOG_STATES;
80
81      typedef enum
82      {
83          /* Application's state machine's initial state. */
84          /* The app mounts the disk */
85          APP_CFG_MOUNT_DISK = 0,
86
87          /* Set the current drive */
88          APP_CFG_SET_CURRENT_DRIVE,
89
90          /* The app opens the file to read */
91          APP_CFG_OPEN_READ_CONFIG_FILE,
92
93          /* The app opens the file to read */
94          APP_CFG_READ_CONFIG_FILE,
95
96          /* The app opens the file to write */
97          APP_CFG_OPEN_WRITE_CONFIG_FILE,
98
99          /* Execute write */
100         APP_CFG_WRITE_CONFIG_FILE,
101
102         /* The app closes the file*/
103         APP_CFG_CLOSE_FILE,
104
105         /* The app closes the file and idles */
106         APP_CFG_IDLE,
107
108         /* An app error has occurred */
109         APP_CFG_ERROR,
110
111         /* Couldnt find config file */
112         APP_CFG_NO_CFG_FILE,
113
114         /* Unmount disk */
115         APP_CFG_UNMOUNT_DISK
116
117     } APP_FAT_CONFIG_STATES;
118
119
120     // *****************************************************************************
121     /* Application Data
122
123       Summary:
124         Holds application data
125
126       Description:
127         This structure holds the application's data.
128
129       Remarks:
130         Application strings and buffers are be defined outside this structure.
131      */
132
133     typedef struct
134     {
135         /* SYS_FS File handle for 1st file */
136         SYS_FS_HANDLE      fileMeasureHandle;
137
138         /* SYS_FS File handle for 2nd file */
139         SYS_FS_HANDLE      fileCfgHandle;
140
141         /* Application's current state */
142         APP_FAT_LOG_STATES         log_state;
143         APP_FAT_CONFIG_STATES      cfg_state;
144
145         /* Application data buffer */
146         char               data[FIFO_RX_SIZE+2] DATA_BUFFER_ALIGN;
```

```
147         /* Application config file */
148         char                cfg_data[200] DATA_BUFFER_ALIGN;
149 
150         /* Filename variable */
151         char                fileName[15] DATA_BUFFER_ALIGN;
152 
153         uint32_t           nBytesWritten;
154 
155         uint32_t           nBytesRead;
156 
157         uint32_t            nBytesToWrite;
158     } APP_FAT_DATA;
159 
160 
161     // *****************************************************************************
162     // *****************************************************************************
163     // Section: Application Callback Routines
164     // *****************************************************************************
165     // *****************************************************************************
166     /* These routines are called by drivers when certain events occur.
167     */
168 
169 
170     // *****************************************************************************
171     // *****************************************************************************
172     // Section: Application Initialization and State Machine Functions
173     // *****************************************************************************
174     // *****************************************************************************
175 
176     /*******************************************************************************
177 
178       Function:
179         void APP_Tasks ( void )
180 
181       Summary:
182         MPLAB Harmony Demo application tasks function
183 
184       Description:
185         This routine is the Harmony Demo application's tasks function.  It
186         defines the application's state machine and core logic.
187 
188       Precondition:
189         The system and application initialization ("SYS_Initialize") should be
190         called before calling this.
191 
192       Parameters:
193         None.
194 
195       Returns:
196         None.
197 
198       Example:
199         <code>
200         APP_Tasks();
201         </code>
202 
203       Remarks:
204         This routine must be called from SYS_Tasks() routine.
205      */
206 
207 
208     void sd_fat_cfg_init(unsigned long *tGnss, unsigned long *tImu, bool *ledState,
        uint32_t *tInactivePeriod);
209 
210     void sd_fat_config_task ( bool init );
211     void sd_CFG_Write (uint32_t tLogGNSS_ms, uint32_t tLogIMU_ms, uint8_t ledState,
        uint32_t tInactiveP, bool skipMount);
212     APP_FAT_CONFIG_STATES sd_cfgGetState( void );
213     void sd_cfgSetState( APP_FAT_CONFIG_STATES newState );
214     char* sd_cfgGetCfgBuffer( void );
215 
216     void sd_fat_logging_task ( void );
217     APP_FAT_LOG_STATES sd_logGetState( void );
```

```
218  void sd_logSetState( APP_FAT_LOG_STATES newState );
219
220  void sd_IMU_scheduleWrite (s_bno055_data * data);
221
222  void sd_GNSS_scheduleWrite (minmea_messages * pGnssData);
223
224  void sd_fat_readDisplayFile(const char * fileName);
225
226
227  #endif /* _APP_H */
228  /*****************************************************************************
229   End of File
230   */
231
232
```

```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
```

```
/**
* @file bno055_support.h
*
*/

/*-------------------------------------------------------------------------*
*  Includes
*-------------------------------------------------------------------------*/
#include "bno055.h"

#define  BNO055_API

#define  FLAG_MEAS_ON    1
#define  FLAG_MEAS_OFF   0
/*-------------------------------------------------------------------------*
*  The following APIs are used for reading and writing of
*   sensor data using I2C communication
*-------------------------------------------------------------------------*/
#ifdef  BNO055_API
#define BNO055_I2C_BUS_WRITE_ARRAY_INDEX ((u8)1)

/*  \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);

/*  \Brief: The API 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 BNO055_I2C_bus_write(u8 dev_addr, u8 reg_addr, u8 *reg_data, u8 cnt);

/*
 * \Brief: I2C init routine
 */
s8 I2C_routine(void);

/*  Brief : The delay routine
 *  \param : delay in ms
 */
void BNO055_delay_msek(u32 msek);

#endif

/*******************End of I2C APIs declarations*********************/

/* This API is an example for reading sensor data
 *  \param: None
 *  \return: communication result
 */
s32 bno055_init_readout(void);

s32 bno055_read_routine(s_bno055_data *data);

/*-------------------------------------------------------------------------*
 *  struct bno055_t parameters can be accessed by using BNO055
 *  BNO055_t having the following parameters
 *  Bus write function pointer: BNO055_WR_FUNC_PTR
 *  Bus read function pointer: BNO055_RD_FUNC_PTR
 *  Burst read function pointer: BNO055_BRD_FUNC_PTR
 *  Delay function pointer: delay_msec
 *  I2C address: dev_addr
```

```
74    *  Chip id of the sensor: chip_id
75    *-------------------------------------------------------------------------*/
76   struct bno055_t bno055;
77
```

```
/* ************************************************************************ */
/** Descriptive File Name

  @Company
    ETML-ES

  @File Name
    mc32_serComm.c

 */
/* ************************************************************************ */

/* ************************************************************************ */
/* ************************************************************************ */
/* Section: Included Files                                                  */
/* ************************************************************************ */
/* ************************************************************************ */
#include "Mc32_serComm.h"
#include <stdio.h>

/* This section lists the other files that are included in this file.
 */

/* TODO:  Include other files here if needed. */


/* ************************************************************************ */
/* ************************************************************************ */
/* Section: File Scope or Global Data                                       */
/* ************************************************************************ */
/* ************************************************************************ */

/*  A brief description of a section can be given directly below the section
    banner.
 */

/* ************************************************************************ */
/** Descriptive Data Item Name

  @Summary
    Brief one-line summary of the data item.

  @Description
    Full description, explaining the purpose and usage of data item.
    <p>
    Additional description in consecutive paragraphs separated by HTML
    paragraph breaks, as necessary.
    <p>
    Type "JavaDoc" in the "How Do I?" IDE toolbar for more information on tags.

  @Remarks
    Any additional remarks
 */


/* ************************************************************************ */
/* ************************************************************************ */
// Section: Local Functions                                                 */
/* ************************************************************************ */
/* ************************************************************************ */



/* ************************************************************************ */
/* ************************************************************************ */
// Section: Interface Functions                                             */
/* ************************************************************************ */
/* ************************************************************************ */

/*  A brief description of a section can be given directly below the section
    banner.
 */

```

```
74  // *****************************************************************************
75
76
77  void serDisplayValues ( s_bno055_data *bno055_data )
78  {
79      char sendBuffer[66] = {0};
80      uint8_t i = 0;
81      static uint32_t ctnTimeout = 0;
82
83      /* Preapare Gravity string */
84      sprintf(sendBuffer, "DT: %d0 ms\tGravity : X = %04.03lf\tY = %04.03lf\tZ =
        %04.03lf \n\r", (bno055_data->d_time), bno055_data->gravity.x, bno055_data->
        gravity.y, bno055_data->gravity.z);
85      /* Transmit Gravity string */
86      do{
87          if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))
88          {
89              PLIB_USART_TransmitterByteSend(USART_ID_1, sendBuffer[i]);
90              i++;
91          }
92          ctnTimeout++;
93      }while((sendBuffer[i-1] != '\r')&&(ctnTimeout<TIME_OUT));
94      i = 0;
95
96      /* Preapare gyroscope string */
97      sprintf(sendBuffer, "Gyro    : X = %04.03lf\tY = %04.03lf\tZ = %04.03lf \n\r",
        bno055_data->gyro.x, bno055_data->gyro.y, bno055_data->gyro.z);
98      /* Transmit Gravity string */
99      do{
100         if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))
101         {
102             PLIB_USART_TransmitterByteSend(USART_ID_1, sendBuffer[i]);
103             i++;
104         }
105         ctnTimeout++;
106     }while((sendBuffer[i-1] != '\r')&&(ctnTimeout<TIME_OUT));
107     i = 0;
108
109     /* Preapare magnitude string */
110     sprintf(sendBuffer, "Mag     : X = %04.03lf\tY = %04.03lf\tZ = %04.03lf \n\r",
        bno055_data->mag.x, bno055_data->mag.y, bno055_data->mag.z);
111     /* Transmit Gravity string */
112     do{
113         if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))
114         {
115             PLIB_USART_TransmitterByteSend(USART_ID_1, sendBuffer[i]);
116             i++;
117         }
118         ctnTimeout++;
119     }while((sendBuffer[i-1] != '\r')&&(ctnTimeout<TIME_OUT));
120     i = 0;
121
122     /* Preapare linear acceleration string */
123     sprintf(sendBuffer, "Accel   : X = %04.03lf\tY = %04.03lf\tZ = %04.03lf \n\r",
        bno055_data->linear_accel.x, bno055_data->linear_accel.y, bno055_data->
        linear_accel.z);
124     /* Transmit Gravity string */
125     do{
126         if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))
127         {
128             PLIB_USART_TransmitterByteSend(USART_ID_1, sendBuffer[i]);
129             i++;
130         }
131         ctnTimeout++;
132     }while((sendBuffer[i-1] != '\r')&&(ctnTimeout<TIME_OUT));
133     i = 0;
134
135     /* Preapare euler string */
136     sprintf(sendBuffer, "Euler   : H = %04.03lf\tP = %04.03lf\tR = %04.03lf \n\r",
        bno055_data->euler.h, bno055_data->euler.p, bno055_data->euler.r);
137     /* Transmit Gravity string */
138     do{
139         if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))
```

```
140             {
141                 PLIB_USART_TransmitterByteSend(USART_ID_1, sendBuffer[i]);
142                 i++;
143             }
144             ctnTimeout++;
145         }while((sendBuffer[i-1] != '\r')&&(ctnTimeout<TIME_OUT));
146         i = 0;
147 
148         /* Preapare quaternion string */
149         sprintf(sendBuffer, "Quater. : W = %05d\tX = %05d\tY = %05d\tZ = %05d \n\n\r",
            bno055_data->quaternion.w, bno055_data->quaternion.x, bno055_data->quaternion.y,
            bno055_data->quaternion.z);
150         /* Transmit Gravity string */
151         do{
152             if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))
153             {
154                 PLIB_USART_TransmitterByteSend(USART_ID_1, sendBuffer[i]);
155                 i++;
156             }
157             ctnTimeout++;
158         }while((sendBuffer[i-1] != '\r')&&(ctnTimeout<TIME_OUT));
159         i = 0;
160 
161     }
162 
163     void serTransmitString ( USART_MODULE_ID usartId, const char * msg )
164     {
165         char bufferMsg[60] = {0};
166         static uint32_t i = 0;
167         static uint32_t ctnTimeout = 0;
168 
169         strncpy(bufferMsg, msg, strlen(msg));
170 
171         /* Transmit string */
172         do{
173             if(!PLIB_USART_TransmitterBufferIsFull(usartId))
174             {
175                 PLIB_USART_TransmitterByteSend(usartId, bufferMsg[i]);
176                 i++;
177             }
178             ctnTimeout++;
179         }while((bufferMsg[i-1] != '\0')&&(ctnTimeout<TIME_OUT));
180         i = 0;
181     }
182 
183     void serTransmitbuffer ( USART_MODULE_ID usartId, char msg[], uint32_t lenght )
184     {
185         uint32_t i = 0;
186         uint32_t ctnTimeout = 0;
187 
188         /* Transmit string */
189         do{
190             if(!PLIB_USART_TransmitterBufferIsFull(usartId))
191             {
192                 PLIB_USART_TransmitterByteSend(usartId, msg[i]);
193                 i++;
194             }
195             ctnTimeout++;
196         }while((i < lenght)&&(ctnTimeout<TIME_OUT));
197         i = 0;
198     }
199 
200     bool pollSerialSingleCmd(USART_MODULE_ID usartID, const char * command1)
201     {
202         static char charRead[30] = {0};
203         static uint32_t readCnt = 0;
204 
205         // Get command's characters
206         while((PLIB_USART_ReceiverDataIsAvailable(usartID))&&(readCnt < 30)){
207             charRead[readCnt] = PLIB_USART_ReceiverByteReceive(usartID);
208             readCnt++;
209         }
210         // Command
```

```
211         if(readCnt >= 30)
212         {
213             /* Reset read counter */
214             readCnt = 0;
215             /* Clear read buffer */
216             memset(charRead,0,strlen(charRead));
217         }
218         // Check occurence with commands
219         if(strstr(charRead, command1) != NULL) {
220             /* Reset read counter */
221             readCnt = 0;
222             /* Clear read buffer */
223             memset(charRead,0,strlen(charRead));
224             /* Command detected */
225             return true;
226         }
227         else{
228             return false;
229         }
230     }
231 
232     bool pollSerialCmds(USART_MODULE_ID usartID, const char * command1, const char *
        command2, const char * command3,
233                         const char * command4)
234     {
235         static char charRead[CHAR_READ_BUFFER_SIZE] = {0};
236         static uint32_t readCnt = 0;
237 
238         // Get command's characters
239         while((PLIB_USART_ReceiverDataIsAvailable(usartID))&&(readCnt <
            CHAR_READ_BUFFER_SIZE)){
240             charRead[readCnt] = PLIB_USART_ReceiverByteReceive(usartID);
241             readCnt++;
242         }
243         // Command
244         if(readCnt >= CHAR_READ_BUFFER_SIZE)
245         {
246             /* Reset read counter */
247             readCnt = 0;
248             /* Clear read buffer */
249             memset(charRead,0,CHAR_READ_BUFFER_SIZE);
250         }
251         // Check occurence with commands
252         if((strstr(charRead, command1) != NULL) || (strstr(charRead, command2) != NULL)
253             || (strstr(charRead, command3) != NULL) || (strstr(charRead, command4) != NULL
                )) {
254             /* Reset read counter */
255             readCnt = 0;
256             /* Clear read buffer */
257             memset(charRead,0,CHAR_READ_BUFFER_SIZE);
258             /* Command detected */
259             return true;
260         }
261         else{
262             return false;
263         }
264     }
265     /* *****************************************************************************
266      End of File
267      */
268 
```

```c
/* ************************************************************************ */
/** Descriptive File Name

  @Company
    ETML-ES

  @File Name
    MC32_serComm.h

 */
/* ************************************************************************ */

#ifndef _SER_COMM_H    /* Guard against multiple inclusion */
#define _SER_COMM_H


/* ************************************************************************ */
/* ************************************************************************ */
/* Section: Included Files                                                  */
/* ************************************************************************ */
/* ************************************************************************ */

#include "app.h"


/* Provide C++ Compatibility */
#ifdef __cplusplus
extern "C" {
#endif


    /* ************************************************************************ */
    /* ************************************************************************ */
    /* Section: Constants                                                       */
    /* ************************************************************************ */
    /* ************************************************************************ */

    /*  A brief description of a section can be given directly below the section
        banner.
     */


    /* ************************************************************************ */
    /** Descriptive Constant Name

      @Summary
        Brief one-line summary of the constant.

      @Description
        Full description, explaining the purpose and usage of the constant.
        <p>
        Additional description in consecutive paragraphs separated by HTML
        paragraph breaks, as necessary.
        <p>
        Type "JavaDoc" in the "How Do I?" IDE toolbar for more information on tags.

      @Remarks
        Any additional remarks
     */
#define EXAMPLE_CONSTANT 0


    // *****************************************************************************
    // *****************************************************************************
    // Section: Data Types
    // *****************************************************************************
    // *****************************************************************************

    /*  A brief description of a section can be given directly below the section
        banner.
     */


```

```
74      // *****************************************************************************
75      // *****************************************************************************
76      // Section: Interface Functions
77      // *****************************************************************************
78      // *****************************************************************************
79
80
81  void serDisplayValues ( s_bno055_data *bno055_data );
82
83  bool pollSerialCmds(USART_MODULE_ID usartID, const char * command1, const char *
84                          command2, const char * command3,const char * command4);
85
86  bool pollSerialSingleCmd(USART_MODULE_ID usartID, const char * command1);
87
88  void serTransmitString ( USART_MODULE_ID usartId, const char * msg );
89  void serTransmitbuffer ( USART_MODULE_ID usartId, char msg[], uint32_t lenght );
90      /* Provide C++ Compatibility */
91  #ifdef __cplusplus
92  }
93  #endif
94
95  #endif /* _EXAMPLE_FILE_NAME_H */
96
97  /* *****************************************************************************
98   End of File
99   */
100
```