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