```
/**-------------------------------------------------------------------------
            \file Monitor.cpp
--                                                                      --
--              ECEN 5803 Mastering Embedded Systems Architecture        --
--                  Project 1 Module 3                                  --
--                Microcontroller Firmware                              --
--                      Monitor.cpp                                       --
--                                                                      --
-------------------------------------------------------------------------
--
--  Designed for:  University of Colorado at Boulder
--
--
--  Designed by:  Tim Scherr
--  Revised by:  David James & Ismail Yesildirek
--
-- Version: 2.0.1
-- Date of current revision:  2018-10-03
-- Target Microcontroller: Freescale MKL25ZVMT4
-- Tools used:  ARM mbed compiler
--              ARM mbed SDK
--              Keil uVision MDK v.5
--              Freescale FRDM-KL25Z Freedom Board
--
--
   Functional Description: See below
--
--      Copyright (c) 2015 Tim Scherr All rights reserved.
--
*/

#include <stdio.h>
#include "shared.h"
DigitalOut greenLED(LED_GREEN); //<! Setup greenLED
bool green_led_status = 1; //<! status variable for greenLED, default is on.

/*****************************************************************************/
/// \fn uint32_t getR0(void)
/// @brief assembly routine which returns the unaltered contents of register r0
/*****************************************************************************/
__asm uint32_t getR0()
{
    BX lr ; returns r0 without altering it
}

/*****************************************************************************/
///  \fn void getRn(uint32_t[16])
/// @brief assembly routine which fills a uint32_t[16] array's 1-15 elements with r1-r15
/*****************************************************************************/
__asm void getRn(uint32_t reglist[16])
{
    STR r1, [r0, #4]  ; r1->reglist[1]
    STR r2, [r0, #4]  ; r2->reglist[2]
    STR r3, [r0, #12] ; r3->reglist[3]
    STR r4, [r0, #16] ; r4->reglist[4]
    STR r5, [r0, #20] ; r5->reglist[5]
    STR r6, [r0, #24] ; r6->reglist[6]
    STR r7, [r0, #28] ; r7->reglist[7]
    MOV r1, r8        ; STR only takes r0-r7
    STR r1, [r0, #32] ; r8->r1->reglist[8]
    MOV r1, r9        ; STR only takes r0-r7
    STR r1, [r0, #36] ; r9->r1->reglist[9]
    MOV r1, r10       ; STR only takes r0-r7
    STR r1, [r0, #40] ; r10->r1->reglist[10]
    MOV r1, r11       ; STR only takes r0-r7
    STR r1, [r0, #44] ; r11->r1->reglist[11]
    MOV r1, r12       ; STR only takes r0-r7
    STR r1, [r0, #48] ; r12->r1->reglist[12]
    MOV r1, sp        ; STR only takes r0-r7
    STR r1, [r0, #52] ; sp(r13)->r1->reglist[13]
    MOV r1, lr        ; STR only takes r0-r7
```

```
72        STR r1, [r0, #56] ; lr(r14)->r1->reglist[14]
73        MOV r1, pc        ; STR only takes r0-r7
74        STR r1, [r0, #60] ; pc(r15)->r1->reglist[15]
75        BX lr             ; return
76    }
77
78    /**************************************************************************/
79    /// @brief uint32_t getWord(uint32_t) assembly routine
80    /// returns the 32 bit word stored at the 32 bit address in the argument
81    /**************************************************************************/
82    __asm uint32_t getWord(uint32_t address)
83    {
84      LDR r0, [r0]  ; data @ r0 -> r0
85      BX lr         ; return
86    }
87
88    /**************************************************************************/
89    ///  \fn void show_regs_and_mem()
90    /// @brief prints registers r0-r15 to debug port,
91    /// and shows the contents of 16 words worth of memory.
92    /// The memory addresses cycle from 0x0 to 0x6000
93    /**************************************************************************/
94    void show_regs_and_mem()
95    {
96      // The top line of the display
97      UART_direct_msg_put("\r\n\r\nRegister contents:");
98      UART_direct_msg_put("\t|\tADDRESS:\tDATA:");
99
100     // retrieve the contents of registers r0-r15
101     uint32_t reg[16]; //!< array to hold r0-r15
102     reg[0] = getR0(); // because passing any variable would alter r0
103     getRn(reg);       // fill the rest of the array
104     uint8_t i=0;      //!< index variable
105
106     // addr is declared static so we can cycle through addresses
107     static uint32_t addr = 0x0; //!< address for memory block reads.
108                                 //< starts at 0x00000000 and increments each iteration
109     uint32_t data;        //<! a variable to hold the data
110     // print 16 lines:
111     for(i=0;i<16;i++)
112     {
113       UART_direct_msg_put("\r\nr"); //start a new line with r
114       //now, to print the register number:
115       if(i/10)  // if the index is greater than 10...
116       { // convert the hex value to decimal
117         UART_direct_hex_put((i/10 << 4) + i%10);
118       }
119       else
120       { // otherwise just print the bottom nibble
121         UART_low_nibble_direct_put(i&0xF);
122       }                                 //add tab for alignment,
123       UART_direct_msg_put(" \t0x");     // format hex values with 0x########
124       UART_direct_word_hex_put(reg[i]); //print the whole word
125       UART_direct_msg_put("\t|\t0x");   // tabs and a divider, plus 0x########
126       UART_direct_word_hex_put(addr);   // first print the address
127       UART_direct_msg_put("\t0x");      // tab and add 0x########
128       data = getWord(addr);             // get the data @ addr
129       UART_direct_word_hex_put(data);   // print the data
130       if(addr < 0x6000) addr +=4;       // increment the address by 4
131       else addr = 0;                    // roll over at 0x6000
132     }
133     return;
134   }
135
136   /**************************************************************************/
137   /// \fn void set_display_mode(void)
138   ///
139   /// Set Display Mode Function
140   /// @brief Function determines the correct display mode.
141   ///
142   ///   The 3 display modes operate as follows:
```

```
143   ///
144   ///  NORMAL MODE       Outputs only mode and state information changes
145   ///                     and calculated outputs
146   ///
147   ///  QUIET MODE        No Outputs
148   ///
149   //  DEBUG MODE        Outputs mode and state information, error counts,
150   ///                     register displays, sensor states, and calculated output
151   ///
152   ///
153   /// There is deliberate delay in switching between modes to allow the RS-232 cable
154   /// to be plugged into the header without causing problems.
155   /*****************************************************************************/
156   void set_display_mode(void)
157   {
158     UART_direct_msg_put("\r\nSelect Mode");
159     UART_direct_msg_put("\r\n Hit NOR - Normal");
160     UART_direct_msg_put("\r\n Hit QUI - Quiet");
161     UART_direct_msg_put("\r\n Hit DEB - Debug" );
162     UART_direct_msg_put("\r\n Hit V - Version#");
163     /****************************************
164      * Added instruction for GREEN LED control
165      * and defined LED under shared.h
166      ****************************************/
167     UART_direct_msg_put("\r\n Hit L - Toggle Green LED\r\n");
168     UART_direct_msg_put("\r\nSelect:  ");
169
170   }
171
172   /*****************************************************************************/
173   /// \fn void chk_UART_msg(void)
174   /// @brief checks for messages in serial port
175   /*****************************************************************************/
176   void chk_UART_msg(void)
177   {
178      UCHAR j; //<! placeholder for character
179      while( UART_input() )      // becomes true only when a byte has been received
180      {                                    // skip if no characters pending
181         j = UART_get();                 // get next character
182
183         if( j == '\r' || j == '\n')          // on a enter (return) key press
184         {                 // complete message (all messages end in carriage return)
185            UART_msg_put("->");
186            UART_msg_process();
187         }
188         else
189         {
190            if ((j != 0x02) )         // if not ^B
191            {                             // if not command, then
192               UART_direct_put(j);              // echo the character
193            }
194            else
195            {
196              ;
197            }
198            if( j == '\b' )
199            {                               // backspace editor
200               if( msg_buf_idx != 0)
201               {                         // if not 1st character then destructive
202                  UART_msg_put(" \b");// backspace
203                  if(msg_buf_idx > 0) msg_buf_idx--;
204               }
205            }
206            else if( msg_buf_idx >= MSG_BUF_SIZE )
207            {                                  // check message length too large
208               UART_msg_put("\r\nToo Long!");
209               msg_buf_idx = 0;
210            }
211            else if ((display_mode == QUIET || display_mode == DEBUG) && (msg_buf[0] != 0x02) &&
212                     (msg_buf[0] != 'D') && (msg_buf[0] != 'N') &&
213                     (msg_buf[0] != 'V') && (msg_buf[0] != 'L') &&
```

```
214                     (msg_buf[0] != 'd') && (msg_buf[0] != 'n') &&
215                     (msg_buf[0] != 'v') && (msg_buf[0] != 'l') &&
216                     (msg_buf_idx != 0))
217             {                           // if first character is bad in Quiet mode
218                msg_buf_idx = 0;         // then start over
219             }
220             else {                         // not complete message, store character
221
222                msg_buf[msg_buf_idx] = j;
223                msg_buf_idx++;
224             }
225          }
226       }
227    }
228
229    /*****************************************************************************/
230    ///  \fn void UART_msg_process(void)
231    /// @brief UART Input Message Processing
232    /*****************************************************************************/
233    void UART_msg_process(void)
234    {
235       UCHAR chr,err=0; //<!   unsigned char for input and error codes;
236
237       chr = msg_buf[0];
238          switch( chr )
239          {
240             case 'D'://Debug mode
241                if((msg_buf[1] == 'E') && (msg_buf[2] == 'B') && (msg_buf_idx == 3))
242                {
243                   display_mode = DEBUG;
244                   UART_msg_put("\r\nMode=DEBUG\n");
245                   display_timer = 0;
246                }
247                else
248                   err = 1;
249                break;
250
251             case 'N': //normal mode
252                if((msg_buf[1] == 'O') && (msg_buf[2] == 'R') && (msg_buf_idx == 3))
253                {
254                   display_mode = NORMAL;
255                   UART_msg_put("\r\nMode=NORMAL\n");
256                   //display_timer = 0;
257                }
258                else
259                   err = 1;
260                break;
261
262             case 'Q': //quiet mode
263                if((msg_buf[1] == 'U') && (msg_buf[2] == 'I') && (msg_buf_idx == 3))
264                {
265                   display_mode = QUIET;
266                   UART_msg_put("\r\nMode=QUIET\n");
267                   display_timer = 0;
268                }
269                else
270                   err = 1;
271                break;
272
273             case 'V': // VERSION;
274                UART_msg_put("\r\n");
275                UART_msg_put( CODE_VERSION );
276                UART_msg_put("\r\nSelect  ");
277                display_timer = 0;
278                break;
279
280             case 'L': //toggle LED;
281                greenLED = !greenLED;
282                green_led_status = !green_led_status;
283                UART_msg_put("\r\n");
284                if (green_led_status ==0){
```

```
285                 UART_msg_put("\r\n Green LED OFF");
286                 }
287                 else
288                 {
289                   UART_msg_put("\r\n Green LED ON");
290                 }
291                 display_timer = 0;
292                 break;
293             //added all lower case input to improve user interface.
294              case 'd':
295                 if((msg_buf[1] == 'e') && (msg_buf[2] == 'b') && (msg_buf_idx == 3))
296                 {
297                    display_mode = DEBUG;
298                    UART_msg_put("\r\nMode=DEBUG\n");
299                    display_timer = 0;
300                 }
301                 else
302                    err = 1;
303                 break;
304
305              case 'n':
306                 if((msg_buf[1] == 'o') && (msg_buf[2] == 'r') && (msg_buf_idx == 3))
307                 {
308                    display_mode = NORMAL;
309                    UART_msg_put("\r\nMode=NORMAL\n");
310                    //display_timer = 0;
311                 }
312                 else
313                    err = 1;
314                 break;
315
316              case 'q':
317                 if((msg_buf[1] == 'u') && (msg_buf[2] == 'i') && (msg_buf_idx == 3))
318                 {
319                    display_mode = QUIET;
320                    UART_msg_put("\r\nMode=QUIET\n");
321                    display_timer = 0;
322                 }
323                 else
324                    err = 1;
325                 break;
326
327              case 'v':
328                 //display_mode = VERSION;
329                 UART_msg_put("\r\n");
330                 UART_msg_put( CODE_VERSION );
331                 //UART_msg_put("\r\nSelect  ");
332                 display_timer = 0;
333                 break;
334
335              case 'l':
336                 greenLED = !greenLED;
337                 green_led_status = !green_led_status;
338                 //display_mode = LED;
339                 UART_msg_put("\r\n");
340                 if (green_led_status ==0){
341                 UART_msg_put("\r\n Green LED OFF");
342                 //UART_msg_put("\r\nSelect  ");
343                 }
344                 else if(green_led_status ==1)
345                 {
346                 UART_msg_put("\r\n Green LED ON");
347                 //UART_msg_put("\r\nSelect  ");
348                 }
349                 display_timer = 0;
350                 break;
351
352            default:
353              err = 1;
354          }
355
```

```
356
357      if( err == 1 )
358      {
359         UART_msg_put("\n\rError!");
360      }
361      else if( err == 2 )
362      {
363         UART_msg_put("\n\rNot in DEBUG Mode!");
364      }
365      else
366      {
367       msg_buf_idx = 0;          // put index to start of buffer for next message
368         ;
369      }
370      msg_buf_idx = 0;          // put index to start of buffer for next message
371
372
373   }
374
375
376   /*****************************************************************************/
377   ///   \fn   is_hex
378   /// @brief Function takes
379   ///  @param a single ASCII character and returns
380   ///  @return 1 if hex digit, 0 otherwise.
381   ///
382   /*****************************************************************************/
383   UCHAR is_hex(UCHAR c)
384   {
385      if( (((c |= 0x20) >= '0') && (c <= '9')) || ((c >= 'a') && (c <= 'f'))  )
386         return 1;
387      return 0;
388   }
389
390   /*****************************************************************************/
391   ///   @brief  DEBUG and DIAGNOSTIC Mode UART Operation
392   /*****************************************************************************/
393   void monitor(void)
394   {
395
396   /*********************************/
397   /*     Spew outputs              */
398   /*********************************/
399
400      switch(display_mode)
401      {
402         case(QUIET):
403            {
404                UART_msg_put("\r\n ");
405                display_flag = 0;
406            }
407            break;
408         case(VERSION):
409            {
410                display_flag = 0;
411            }
412            break;
413         case(LED):
414            {
415              display_flag = 0;
416            }
417            break;
418         case(NORMAL):
419            {
420               if (display_flag == 1)
421               {
422                  UART_msg_put("\r\nNORMAL ");
423                  UART_msg_put(" Flow: ");
424                  // ECEN 5803 add code as indicated
425                  //  add flow data output here, use UART_hex_put or similar for
426                  // numbers
```

```
427                 UART_hex_put('2');
428                 UART_msg_put(" Temp: ");
429                 //  add flow data output here, use UART_hex_put or similar for
430                 // numbers
431                 UART_hex_put('7');
432                 UART_msg_put(" Freq: ");
433                 //  add flow data output here, use UART_hex_put or similar for
434                 // numbers
435                 UART_hex_put('5');
436                 display_flag = 0;
437             }
438          }
439          break;
440       case(DEBUG):
441          {
442             if (display_flag == 1)
443             {
444                UART_msg_put("\r\nDEBUG ");
445                UART_msg_put(" Flow: ");
446                // ECEN 5803 add code as indicated
447                //  add flow data output here, use UART_hex_put or similar for
448                // numbers
449                UART_hex_put('2');
450                UART_msg_put(" Temp: ");
451                //  add flow data output here, use UART_hex_put or similar for
452                // numbers
453                UART_hex_put('7');
454                UART_msg_put(" Freq: ");
455                //  add flow data output here, use UART_hex_put or similar for
456                // numbers
457                UART_hex_put('5');
458
459    /****************      ECEN 5803 add code as indicated   ***************/
460                //  Create a display of  error counts, sensor states, and
461                //  ARM Registers R0-R15
462                //UART_msg_put("\r\n Error: ");
463                //UART_msg_put("\r\n Sensor: ");
464
465                show_regs_and_mem(); // function displays register contents over UART
466
467                //  Create a command to read 16 words from the current stack
468                // and display it in reverse chronological order.
469
470
471                // clear flag to ISR
472                display_flag = 0;
473              }
474          }
475          break;
476
477       default:
478       {
479          UART_msg_put("Mode Error");
480       }
481     }
482   }
483
```