```
2
           \file UART poll.cpp
3
                    ECEN 5803 Mastering Embedded System Architecture
                       Project 1 Module 3
7
                      Microcontroller Firmware
8
                            UART poll.c
9
10
11
12
        Designed for: University of Colorado at Boulder
13
14
15
    -- Designed by: Tim Scherr
    -- Revised by: David James & Ismail Yesildirek
17
18
    -- Version: 2.0.1
    -- Date of current revision: 2018-10-03
19
20
    -- Target Microcontroller: Freescale MKL25ZVMT4
    -- Tools used: ARM mbed compiler
21
22
                    ARM mbed SDK
23
                    Keil uVision MDK v.5
24
                    Freescale FRDM-KL25Z Freedom Board
25
26
27 -- Functional Description: This file contains routines that support messages
         to and from the UART port. Included are:
29
    ___
            Serial() - a routine to send/receive bytes on the UART port to
30
    ___
                           the transmit/receive buffers
            UART put() - a routine that puts a character in the transmit buffer
31
    ___
32
    --
            UART get() - a routine that gets the next character from the receive
                           buffer
33
    --
34
            UART_msg_put() - a routine that puts a string in the transmit buffer
35
             UART direct msg put() - routine that sends a string out the UART port
36
             UART input() - determines if a character has been received
37
             UART_hex_put() - a routine that puts a hex byte in the transmit buffer
38
            UART_direct_hex_put - a routine that puts a hex byte directly to the UART port
39
40
            NEW TO VERSION 2.0.1:
41
             UART_low_nibble_direct_put() - puts the low nibble of a byte in hex directly
42
                                             (no ram buffer) to the UART.
             UART direct_word_hex_put() - puts a word in hex directly to the UART
43
            Copyright (c) 2015 Tim Scherr All rights reserved.
    --
44
4.5
46
   */
47
48
49
   /*******
50
   /* Configurations */
51
52
    /*
53
54
55
56
57
    #include <stdio.h>
58
    #include "shared.h"
    #include "MKL25Z4.h"
59
    // NOTE: UARTO is also called UARTLP in mbed
61
    #define OERR (UARTO->S1 & UARTLP S1 OR MASK)
                                                  // Overrun Error bit
62
    #define CREN (UARTO->C2 & UARTLP C2 RE MASK)
63
                                                  // continuous receive enable bit
                                                  // Receive Data Register
    #define RCREG UART0->D
64
                                                 // Framing Error bit
65
    #define FERR (UARTO->S1 & UARTLP S1 FE MASK)
    #define RCIF (UARTO->S1 & UARTLP S1 RDRF MASK) // Receive Interrupt Flag (full)
66
    #define TXIF (UART0->S1 & UARTLP_S1_TDRE_MASK) // Transmit Interrupt Flag (empty)
67
                                                 // Transmit Data Register
68
    #define TXREG UART0->D
69
    #define TRMT (UARTO->S1 & UARTLP S1 TC MASK)
                                                 // Transmit Shift Register Empty
70
    /**********
```

```
Start of code
 73
 74
 7.5
      UCHAR error_count = 0;
 76
 77
     /// \fn void serial(void)
 78
     /// function polls the serial port for Rx or Tx data
                            // The serial function polls the serial port for
 79
     void serial(void)
 80
                             // received data or data to transmit
 81
 82
                              // deals with error handling first
 83
        if ( OERR )
                              // if an overrun error, clear it and continue.
 84
         {
 85
           error count++;
 86
                                 // resets and sets continous receive enable bit
 87
           UARTO->C2 = UARTO->C2 & (!UARTLP C2 RE MASK);
 88
           UARTO->C2 = UARTO->C2 | UARTLP C2 RE MASK;
 89
 90
 91
        if (FERR) {
                          // if a framing error, read bad byte, clear it and continue.
 92
           error count++;
 93
                          // This will also clear RCIF if only one byte has been
           RCREG;
 94
                          // received since the last int, which is our assumption.
 95
 96
                          // resets and sets continous receive enable bit
 97
           UARTO -> C2 = UARTO -> C2 & (!UARTLP C2 RE MASK);
           UARTO->C2 = UARTO->C2 | UARTLP C2 RE MASK;
 99
        }
100
        else
                          // else if no frame error,
101
         {
           if ( RCIF )
102
                       // Check if we have received a byte
103
                         // Read byte to enable reception of more bytes
104
                         // For PIC, RCIF automatically cleared when RCREG is read
105
                         // Also true of Freescale KL25Z
106
              *rx in ptr++ = RCREG;
                                       /* get received character */
107
              if( rx in ptr >= RX BUF SIZE + rx buf )
108
109
                 110
                                          to top of buffer */
111
              }
112
113
114
        }
115
                          // Check if transmit buffer empty
116
        if (TXIF)
117
           if ((tx_in_ptr != tx_out_ptr) && (display_mode != QUIET))
118
119
120
              TXREG = *tx out ptr++;
                                       /* send next char */
              if( tx out ptr >= TX BUF SIZE + tx buf )
121
122
                 tx out ptr = tx buf;
                                              /* 0 <= tx out idx < TX BUF SIZE */
123
              tx in progress = true;
                                            /* flag needed to start up after idle */
124
           }
125
           else
126
           {
                                                /* no more to send */
127
              tx in progress = false;
128
129
130
     // serial count++;
                                // increment serial counter, for debugging only
131
      serial \overline{flag} = 1;
                              // and set flag
132
     /*************************
133
     * The function UART direct msg put puts a null terminated string directly
134
     ^{\star} (no ram buffer) to the UART in ASCII format.
135
136
     void UART TX wait()
137
138
139
       while( TXIF == 0 );
140
141
142
```

```
/**********************
144
     * The function UART direct msg put puts a null terminated string directly
145
     * (no ram buffer) to the UART in ASCII format.
146
147
     void UART_direct_msg_put(const char *str)
148
149
        while( *str != '\0')
150
151
          TXREG = *str++;
152
          while( TXIF == 0 || TRMT == 0 ); // waits here for UART transmit buffer
153
                                       // to be empty
154
155
    }
156
     /****************************
157
158
     * The function UART put puts a byte, to the transmit buffer at the location
159
     * pointed to by tx in idx. The pointer is incremented circularly as described
160
     * previously. If the transmit buffer should wrap around (should be designed
     * not to happen), data will be lost. The serial interrupt must be temporarily
161
     * disabled since it reads tx in idx and this routine updates tx in idx which is
162
     * a 16 bit value.
163
                      ************************************
164
165
166
     void UART put(UCHAR c)
167
168
        *tx in ptr++ = c;
                                        // save character to transmit buffer
        if( tx in ptr >= TX BUF SIZE + tx buf)
          tx in ptr = tx buf;
170
                                               // 0 <= tx in idx < TX BUF SIZE
171
172
     /****************************
173
     * The function UART direct_put puts a ncharacter directly
174
175
     * (no ram buffer) to the UART in ASCII format.
     ******************************
176
177
     void UART direct put(UCHAR c)
178
179
        TXREG = c;
180
        UART_TX_wait();
181
     /****************************
182
     * The function UART get gets the next byte if one is available from the receive
183
     * buffer at the location pointed to by rx out idx. The pointer is circularly
184
     * incremented and the byte is returned in R7. Should no byte be available the
185
     ^{\star} function will wait until one is available. There is no need to disable the
186
     ^{\star} serial interrupt which modifies rx_in_idx since the function is looking for a
187
188
     * compare only between rx_in_idx & rx_out_idx.
189
190
     UCHAR UART get(void)
191
192
        UCHAR c;
193
        while( rx in ptr == rx out ptr );  /* wait for a received character,
194
                                                              indicated */
195
                                            // when pointers are different
196
                                           // this could be an infinite loop, but
197
                                           // is not because of UART input check
        c = *rx_out_ptr++;
198
199
        if( rx out ptr >= RX BUF SIZE + rx buf ) // if at end of buffer
200
          rx_out_ptr = rx buf;
201
                                           /* 0 <= rx out idx < RX BUF SIZE */
202
                                          // return byte from beginning of buffer
203
                                         // next time.
204
        return(c);
205
206
207
     /**********************************
208
     * The function UART input returns a 1 if 1 or more receive byte(s) is(are)
209
     ^{\star} available and a 0 if the receive buffer rx_buf is empty. There is no need to
210
     * disable the serial interrupt which modifies rx in idx since function is
211
212
     * looking for a compare only between rx in idx & rx out idx.
213
```

C:\Users\David James\Documents\KeilProjects\MESA\Mod4 (2)\Mod4\UART_poll.cpp

```
UCHAR UART input (void)
215
216
       if( rx in ptr == rx out ptr )return(0);
217
                           /* no characters in receive buffer */
218
        return(1);
                                   /* 1 or more receive characters ready */
219
220
    221
    * The function UART msg put puts a null terminated string through the transmit
222
223
    * buffer to the UART port in ASCII format.
    ************************
224
225
226
    void UART_msg_put(const char *str)
227
228
      while( *str != '\0')
229
         *tx in ptr++ = *str++; // save character to transmit buffer
230
231
         if (tx in ptr >= TX BUF SIZE + tx buf)
232
           tx in ptr = tx buf;
                                        // 0 <= tx in idx < TX BUF SIZE</pre>
233
234
    * /
235
    /******************************
236
    * HEX TO ASC Function
237
238
    * Function takes a single hex character (0 thru Fh) and converts to ASCII.
239
240
    UCHAR hex to asc(UCHAR c)
241
242
    if( c <= 9 )
243
        return(c + 0x30);
      return( ((c & 0x0f) + 0x37 )); /* add 37h */
244
245
246
    /****************************
247
248
    * ASC TO HEX Function
249
    * Function takes a single ASCII character and converts to hex.
250
251
252
    UCHAR asc to hex(UCHAR c)
253
      if( c <= '9')
254
255
        return ( c - 0x30 );
      return( (c & 0xdf) - 0x37 ); // clear bit 5 (lower case) & subtract 37h
256
257
    * /
258
259
    /****************************
260
261
    * The function UART low nibble put puts the low nibble of a byte in hex directly
    * (no ram buffer) to the UART.
262
                           263
264
    void UART low nibble direct put(UCHAR c)
265
266
      TXREG = hex to asc( c & 0x0f);
267
      UART_TX_wait();
268
269
    /******************************
270
271
    * The function UART high nibble put puts the high nibble of a byte in h
272
    * UART port.
                *******************
273
274
    //void UART high nibble put(unsigned char c)
275
276
        UART put( hex to asc( (c>>4) & 0x0f ));
277
    //}
278
279
    * The function UART_hex_put puts 1 byte in hex through the transmit buffer to
280
281
    * the UART port.
             282
283
284
    void UART hex put(unsigned char c)
```

C:\Users\David James\Documents\KeilProjects\MESA\Mod4 (2)\Mod4\UART_poll.cpp

```
285
286
       UART put( hex to asc( (c>>4) & 0x0f )); // could eliminate & as >> of UCHAR
287
                                           // by definition clears upper bits.
       UART put( hex to asc( c & 0x0f ));
288
289
290
     /*****************************
291
     * The function UART direct hex put puts 1 byte in hex directly (no ram buffer)
292
293
    * to the UART.
     *****************************
294
    void UART_direct_hex_put(unsigned char c)
295
296
297
       TXREG = hex_to_asc((c>>4) & 0x0f);
298
       UART_TX_wait();
299
       TXREG = hex to asc( c & 0x0f );
300
       UART_TX_wait();
301
     }
     /*****************************
302
    * The function UART direct hex put puts 4 bytes in hex directly (no ram buffer)
303
     * to the UART.
304
                     ***************
305
306
     void UART direct word hex put(uint32 t word)
307
308
     UART direct hex put((word>>24)&0xFF);
309
     UART TX wait();
310
     UART direct hex put((word>>16)&0xFF);
     UART TX wait();
311
     UART direct hex put((word>>8)&0xFF);
312
313
     UART TX wait();
314
     UART direct hex put(word&0xFF);
315
      UART TX wait();
316
317
318
    * The function UART direct hex int put removes the leading zeroes and adds
     a decimal point. Best used in conjunction with hex2hexInt
319
    *************************
320
321
    void UART_direct_hex_int_put(uint32_t word, uint8_t deci)
322
323
      bool zeros = true;
324
      int8 t i = 7; //must be signed because of wrap-around
       for (i=7; i>=0; i--)
325
326
327
        if (zeros && ((word>>(i*4))&0xF)==0);
328
        else
329
330
         zeros = false;
         UART low_nibble_direct_put((word>>i*4)&0xF);
331
332
          if(i == deci & deci>0) UART direct put('.');
333
334
335
     }
336
```