```
/**-------------------------------------------------------------------------
      \file UART_poll.cpp

--                                                                          --
--              ECEN 5803 Mastering Embedded System Architecture            --
--                  Project 1 Module 3                                      --
--                Microcontroller Firmware                                  --
--                      UART_poll.c                                         --
--                                                                          --
----------------------------------------------------------------------------
--
--  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:  This file contains routines that support messages
--    to and from the UART port.  Included are:
--       Serial() - a routine to send/receive bytes on the UART port to
--                       the transmit/receive buffers
--       UART_put()  - a routine that puts a character in the transmit buffer
--       UART_get()  - a routine that gets the next character from the receive
--                       buffer
--       UART_msg_put() - a routine that puts a string in the transmit buffer
--       UART_direct_msg_put() - routine that sends a string out the UART port
--       UART_input() - determines if a character has been received
--       UART_hex_put() - a routine that puts a hex byte in the transmit buffer
--       UART_direct_hex_put - a routine that puts a hex byte directly to the UART port
--
--      NEW TO VERSION 2.0.1:
--        UART_low_nibble_direct_put() - puts the low nibble of a byte in hex directly
--                                       (no ram buffer) to the UART.
--        UART_direct_word_hex_put() - puts a word in hex directly to the UART
--      Copyright (c) 2015 Tim Scherr  All rights reserved.
--
*/



/*******************/
/*  Configurations */
/*******************/
/*

*/

#include <stdio.h>
#include "shared.h"
#include "MKL25Z4.h"

// NOTE:  UART0 is also called UARTLP in mbed
#define OERR (UART0->S1 & UARTLP_S1_OR_MASK)    // Overrun Error bit
#define CREN (UART0->C2 & UARTLP_C2_RE_MASK)    // continuous receive enable bit
#define RCREG UART0->D                          // Receive Data Register
#define FERR (UART0->S1 & UARTLP_S1_FE_MASK)    // Framing Error bit
#define RCIF (UART0->S1 & UARTLP_S1_RDRF_MASK) // Receive Interrupt Flag (full)
#define TXIF (UART0->S1 & UARTLP_S1_TDRE_MASK) // Transmit Interrupt Flag (empty)
#define TXREG UART0->D                          // Transmit Data Register
#define TRMT (UART0->S1 & UARTLP_S1_TC_MASK)    // Transmit Shift Register Empty

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

```
 72    *        Start of code            *
 73    *********************************/
 74
 75    UCHAR error_count = 0; //<! variable for error counter
 76   /*************************************************************************/
 77   ///  \fn void serial(void)
 78   /// @brief function polls the serial port for Rx or Tx data
 79   /*************************************************************************/
 80   void serial(void)       // The serial function polls the serial port for
 81                           // received data or data to transmit
 82   {
 83                            // deals with error handling first
 84      if ( OERR )           // if an overrun error, clear it and continue.
 85      {
 86         error_count++;
 87                               // resets and sets continous receive enable bit
 88         UART0->C2 = UART0->C2 & (!UARTLP_C2_RE_MASK);
 89         UART0->C2 = UART0->C2 | UARTLP_C2_RE_MASK;
 90      }
 91
 92      if ( FERR){       // if a framing error, read bad byte, clear it and continue.
 93         error_count++;
 94         RCREG;         // This will also clear RCIF if only one byte has been
 95                        // received since the last int, which is our assumption.
 96
 97                        // resets and sets continous receive enable bit
 98         UART0->C2 = UART0->C2 & (!UARTLP_C2_RE_MASK);
 99         UART0->C2 = UART0->C2 | UARTLP_C2_RE_MASK;
100      }
101      else              // else if no frame error,
102      {
103         if ( RCIF )    // Check if we have received a byte
104         {              // Read byte to enable reception of more bytes
105                        // For PIC, deb
106           //RCIF automatically cleared when RCREG is read
107                        // Also true of Freescale KL25Z
108           *rx_in_ptr++ = RCREG;         /* get received character */
109           if( rx_in_ptr >= RX_BUF_SIZE + rx_buf )
110           {
111              rx_in_ptr = rx_buf;    /* if at end of buffer, circles rx_in_ptr
112                                        to top of buffer */
113           }
114
115         }
116      }
117
118      if (TXIF)          //  Check if transmit buffer empty
119      {
120         if ((tx_in_ptr != tx_out_ptr) && (display_mode != QUIET))
121         {
122            TXREG = *tx_out_ptr++;     /* send next char */
123            if( tx_out_ptr >= TX_BUF_SIZE + tx_buf )
124               tx_out_ptr = tx_buf;          /* 0 <= tx_out_idx < TX_BUF_SIZE */
125            tx_in_progress = YES;          /* flag needed to start up after idle */
126         }
127         else
128         {
129            tx_in_progress = NO;             /* no more to send */
130         }
131     }
132   //  serial_count++;         // increment serial counter, for debugging only
133     serial_flag = 1;        // and set flag
134   }
135
136   /*************************************************************************/
137   /// @brief  UART_direct_msg_put puts a null terminated string directly
138   /// (no ram buffer) to the UART in ASCII format.
139   /*************************************************************************/
140   void UART_direct_msg_put(const char *str)
141   {
142      while( *str != '\0' )
```

```
143      {
144         TXREG = *str++;
145         while( TXIF == 0 || TRMT == 0 );  // waits here for UART transmit buffer
146      }
147   }
148
149   /*****************************************************************************/
150   /// @brief UART_direct_msg_put puts a character directly
151   /// (no ram buffer) to the UART in ASCII format.
152   /*****************************************************************************/
153   void UART_direct_put(UCHAR chr)
154   {
155         TXREG = chr;
156         while( TXIF == 0 )
157      {
158       //  __clear_watchdog_timer();
159      }
160   }
161
162   /*****************************************************************************/
163   /// \fn UART_put(UCHAR)
164   /// @brief Puts a byte into the transmit buffer
165   ///
166   /// Puts a byte, to the transmit buffer at the location
167   /// pointed to by tx_in_idx.  The pointer is incremented circularly as described
168   /// previously.  If the transmit buffer should wrap around (should be designed
169   /// not to happen), data will be lost.  The serial interrupt must be temporarily
170   /// disabled since it reads tx_in_idx and this routine updates tx_in_idx which is
171   /// a 16 bit value.
172   /*****************************************************************************/
173   void UART_put(UCHAR c)
174   {
175      *tx_in_ptr++ = c;                     /* save character to transmit buffer */
176      if( tx_in_ptr >= TX_BUF_SIZE + tx_buf)
177         tx_in_ptr = tx_buf;                     /* 0 <= tx_in_idx < TX_BUF_SIZE */
178   }
179
180   /*****************************************************************************/
181   /// \fn UART_get
182   /// @brief Gets the next byte available in the RX buffer.
183
184   /// UART_get gets the next byte if one is available from the receive
185   /// buffer at the location pointed to by rx_out_idx.  The pointer is circularly
186   /// incremented and the byte is returned in R7. Should no byte be available the
187   /// function will wait until one is available. There is no need to disable the
188   /// serial interrupt which modifies rx_in_idx since the function is looking for a
189   // compare only between rx_in_idx & rx_out_idx.
190   /*****************************************************************************/
191   UCHAR UART_get(void)
192   {
193      UCHAR c;
194      while( rx_in_ptr == rx_out_ptr );      /* wait for a received character,
195                                                              indicated */
196                                             // when pointers are different
197                                             // this could be an infinite loop, but
198                                             // is not because of UART_input check
199      c = *rx_out_ptr++;
200      if( rx_out_ptr >= RX_BUF_SIZE + rx_buf )  // if at end of buffer
201      {
202         rx_out_ptr = rx_buf;                 /* 0 <= rx_out_idx < RX_BUF_SIZE */
203                                             // return byte from beginning of buffer
204      }                                      // next time.
205      return(c);
206   }
207
208   /*****************************************************************************/
209   /// @brief Checks if characters are sitting in the UART input buffer.
210
211   ///
212   /// UART_input returns a 1 if 1 or more receive byte(s) is(are)
213   /// available and a 0 if the receive buffer rx_buf is empty.  There is no need to
```

```
214   /// disable the serial interrupt which modifies rx_in_idx since function is
215   /// looking for a compare only between rx_in_idx & rx_out_idx.
216   /*****************************************************************************/
217   UCHAR UART_input(void)
218   {
219      if( rx_in_ptr == rx_out_ptr )
220         return(0);                          /* no characters in receive buffer */
221      else
222         return(1);                         /* 1 or more receive characters ready */
223   }
224
225   /*****************************************************************************/
226   /// @brief UART_msg_put puts a null terminated string through the transmit
227   /// buffer to the UART port in ASCII format.
228   /*****************************************************************************/
229   void UART_msg_put(const char *str)
230   {
231      while( *str != '\0' )
232      {
233         *tx_in_ptr++ = *str++;         /* save character to transmit buffer */
234         if( tx_in_ptr >= TX_BUF_SIZE + tx_buf)
235            tx_in_ptr = tx_buf;                  /* 0 <= tx_in_idx < TX_BUF_SIZE */
236      }
237   }
238
239   /*****************************************************************************/
240   /// @brief HEX_TO_ASCII Function
241   /// Function takes a single hex character (0 thru Fh) and converts to ASCII.
242   /*****************************************************************************/
243   UCHAR hex_to_asc(UCHAR c)
244   {
245      if( c <= 9 )
246         return( c + 0x30 );
247      return( ((c & 0x0f) + 0x37 ));         /* add 37h */
248   }
249
250   /*****************************************************************************/
251   /// @brief Takes a single ASCII character and converts to hex.
252   /*****************************************************************************/
253   UCHAR asc_to_hex(UCHAR c)
254   {
255      if( c <= '9' )
256         return( c - 0x30 );
257      return( (c & 0xdf) - 0x37 );    /* clear bit 5 (lower case) & subtract 37h */
258   }
259
260
261   /*****************************************************************************/
262   /// @brief UART_low_nibble_put puts the low nibble of a byte in hex directly
263   /// (no ram buffer) to the UART.
264   /*****************************************************************************/
265   void UART_low_nibble_direct_put(UCHAR c)
266   {
267      TXREG = hex_to_asc( c & 0x0f );
268         while( TXIF == 0 );
269   }
270
271   /*****************************************************************************/
272   /// @brief UART_high_nibble_put puts the high nibble of a byte in h
273   /// UART port. (currently commented out)
274   /*****************************************************************************/
275   //void UART_high_nibble_put(unsigned char c)
276   //{
277   //   UART_put( hex_to_asc( (c>>4) & 0x0f ));
278   //}
279
280   /*****************************************************************************
281   *! \brief UART_hex_put puts 1 byte in hex through the transmit buffer to
282   * the UART port.
283   *****************************************************************************/
284   void UART_hex_put(unsigned char c)
```

```
285   {
286      UART_put( hex_to_asc( (c>>4) & 0x0f ));  // could eliminate & as >> of UCHAR
287                                                // by definition clears upper bits.
288      UART_put( hex_to_asc( c & 0x0f ));
289   }
290
291   /*****************************************************************************/
292   /// @brief UART_direct_hex_put puts 1 byte in hex directly (no ram buffer)
293   /// to the UART.
294   /*****************************************************************************/
295   void UART_direct_hex_put(unsigned char c)
296   {
297      TXREG = hex_to_asc( (c>>4) & 0x0f );
298      while( TXIF == 0 )
299      {
300       //  __clear_watchdog_timer();
301      }
302      TXREG = hex_to_asc( c & 0x0f );
303      while( TXIF == 0 )
304      {
305       //  __clear_watchdog_timer();
306      }
307   }
308   /*****************************************************************************/
309   /// @brief UART_direct_hex_put puts 4 bytes in hex directly (no ram buffer)
310   /// to the UART.
311   /*****************************************************************************/
312   void UART_direct_word_hex_put(uint32_t word)
313   {
314     UART_direct_hex_put((word>>24)&0xFF);
315     UART_direct_hex_put((word>>16)&0xFF);
316     UART_direct_hex_put((word>>8)&0xFF);
317     UART_direct_hex_put(word&0xFF);
318   }
319
```