```
/**-----------------------------------------------------------------------------
       \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

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

```
 *        Start of code            *
 ***********************************/

 UCHAR error_count = 0;

///  \fn void serial(void)
/// function polls the serial port for Rx or Tx data
void serial(void)       // The serial function polls the serial port for
                        // received data or data to transmit
{
                         // deals with error handling first
   if ( OERR )           // if an overrun error, clear it and continue.
   {
      error_count++;
                            // resets and sets continous receive enable bit
      UART0->C2 = UART0->C2 & (!UARTLP_C2_RE_MASK);
      UART0->C2 = UART0->C2 | UARTLP_C2_RE_MASK;
   }

   if ( FERR){       // if a framing error, read bad byte, clear it and continue.
      error_count++;
      RCREG;         // This will also clear RCIF if only one byte has been
                     // received since the last int, which is our assumption.

                     // resets and sets continous receive enable bit
      UART0->C2 = UART0->C2 & (!UARTLP_C2_RE_MASK);
      UART0->C2 = UART0->C2 | UARTLP_C2_RE_MASK;
   }
   else              // else if no frame error,
   {
      if ( RCIF )   // Check if we have received a byte
      {             // Read byte to enable reception of more bytes
                    // For PIC, RCIF automatically cleared when RCREG is read
                    // Also true of Freescale KL25Z
         *rx_in_ptr++ = RCREG;         /* get received character */
         if( rx_in_ptr >= RX_BUF_SIZE + rx_buf )
         {
            rx_in_ptr = rx_buf;    /* if at end of buffer, circles rx_in_ptr
                                      to top of buffer */
         }

      }
   }

   if (TXIF)          //  Check if transmit buffer empty
   {
      if ((tx_in_ptr != tx_out_ptr) && (display_mode != QUIET))
      {
         TXREG = *tx_out_ptr++;     /* send next char */
         if( tx_out_ptr >= TX_BUF_SIZE + tx_buf )
            tx_out_ptr = tx_buf;           /* 0 <= tx_out_idx < TX_BUF_SIZE */
         tx_in_progress = true;          /* flag needed to start up after idle */
      }
      else
      {
         tx_in_progress = false;             /* no more to send */
      }
  }
//  serial_count++;         // increment serial counter, for debugging only
  serial_flag = 1;        // and set flag
}
/*******************************************************************************
* The function UART_direct_msg_put puts a null terminated string directly
* (no ram buffer) to the UART in ASCII format.
******************************************************************************/
void UART_TX_wait()
{
  while( TXIF == 0 );
}
```

```
/*****************************************************************************
* The function UART_direct_msg_put puts a null terminated string directly
* (no ram buffer) to the UART in ASCII format.
*****************************************************************************/
void UART_direct_msg_put(const char *str)
{
   while( *str != '\0' )
   {
      TXREG = *str++;
      while( TXIF == 0 || TRMT == 0 ); // waits here for UART transmit buffer
                                      // to be empty
   }
}

/*****************************************************************************
* The function UART_put puts a byte, to the transmit buffer at the location
* pointed to by tx_in_idx.  The pointer is incremented circularly as described
* previously.  If the transmit buffer should wrap around (should be designed
* not to happen), data will be lost.  The serial interrupt must be temporarily
* disabled since it reads tx_in_idx and this routine updates tx_in_idx which is
* a 16 bit value.
*****************************************************************************/
/*
void UART_put(UCHAR c)
{
   *tx_in_ptr++ = c;                    // save character to transmit buffer
   if( tx_in_ptr >= TX_BUF_SIZE + tx_buf)
      tx_in_ptr = tx_buf;                     // 0 <= tx_in_idx < TX_BUF_SIZE
}
*/
/*****************************************************************************
* The function UART_direct_put puts a ncharacter directly
* (no ram buffer) to the UART in ASCII format.
*****************************************************************************/
void UART_direct_put(UCHAR c)
{
   TXREG = c;
   UART_TX_wait();
}
/*****************************************************************************
* The function UART_get gets the next byte if one is available from the receive
* buffer at the location pointed to by rx_out_idx.  The pointer is circularly
* incremented and the byte is returned in R7. Should no byte be available the
* function will wait until one is available. There is no need to disable the
* serial interrupt which modifies rx_in_idx since the function is looking for a
* compare only between rx_in_idx & rx_out_idx.
*****************************************************************************/
UCHAR UART_get(void)
{
   UCHAR c;
   while( rx_in_ptr == rx_out_ptr );    /* wait for a received character,
                                                                indicated */
                                          // when pointers are different
                                          // this could be an infinite loop, but
                                          // is not because of UART_input check
   c = *rx_out_ptr++;
   if( rx_out_ptr >= RX_BUF_SIZE + rx_buf )  // if at end of buffer
   {
      rx_out_ptr = rx_buf;                /* 0 <= rx_out_idx < RX_BUF_SIZE */
                                        // return byte from beginning of buffer
   }                                    // next time.
   return(c);

}

/*****************************************************************************
* The function UART_input returns a 1 if 1 or more receive byte(s) is(are)
* available and a 0 if the receive buffer rx_buf is empty.  There is no need to
* disable the serial interrupt which modifies rx_in_idx since function is
* looking for a compare only between rx_in_idx & rx_out_idx.
*****************************************************************************/
```

```
UCHAR UART_input(void)
{
   if( rx_in_ptr == rx_out_ptr )return(0);
                              /* no characters in receive buffer */
      return(1);                        /* 1 or more receive characters ready */
}

/*******************************************************************************
* The function UART_msg_put puts a null terminated string through the transmit
* buffer to the UART port in ASCII format.
*******************************************************************************/
/*
void UART_msg_put(const char *str)
{
   while( *str != '\0' )
   {
      *tx_in_ptr++ = *str++;        // save character to transmit buffer
      if( tx_in_ptr >= TX_BUF_SIZE + tx_buf)
         tx_in_ptr = tx_buf;                   // 0 <= tx_in_idx < TX_BUF_SIZE
   }
}
*/
/*******************************************************************************
* HEX_TO_ASC Function
* Function takes a single hex character (0 thru Fh) and converts to ASCII.
*******************************************************************************/
UCHAR hex_to_asc(UCHAR c)
{
   if( c <= 9 )
      return( c + 0x30 );
   return( ((c & 0x0f) + 0x37 ));        /* add 37h */
}

/*******************************************************************************
* ASC_TO_HEX Function
* Function takes a single ASCII character and converts to hex.
*******************************************************************************/
/*
UCHAR asc_to_hex(UCHAR c)
{
   if( c <= '9' )
      return( c - 0x30 );
   return( (c & 0xdf) - 0x37 );    // clear bit 5 (lower case) & subtract 37h
}
*/

/*******************************************************************************
* The function UART_low_nibble_put puts the low nibble of a byte in hex directly
*  (no ram buffer) to the UART.
*******************************************************************************/
void UART_low_nibble_direct_put(UCHAR c)
{
   TXREG = hex_to_asc( c & 0x0f );
   UART_TX_wait();
}

/*******************************************************************************
* The function UART_high_nibble_put puts the high nibble of a byte in h
* UART port.
*******************************************************************************/
//void UART_high_nibble_put(unsigned char c)
//{
//   UART_put( hex_to_asc( (c>>4) & 0x0f ));
//}

/*******************************************************************************
* The function UART_hex_put puts 1 byte in hex through the transmit buffer to
* the UART port.
*******************************************************************************/
/*
void UART_hex_put(unsigned char c)
```

```
{
   UART_put( hex_to_asc( (c>>4) & 0x0f ));  // could eliminate & as >> of UCHAR
                                             // by definition clears upper bits.
   UART_put( hex_to_asc( c & 0x0f ));
}
*/
/*******************************************************************************
* The function UART_direct_hex_put puts 1 byte in hex directly (no ram buffer)
* to the UART.
*******************************************************************************/
void UART_direct_hex_put(unsigned char c)
{
   TXREG = hex_to_asc( (c>>4) & 0x0f );
   UART_TX_wait();
   TXREG = hex_to_asc( c & 0x0f );
   UART_TX_wait();
}
/*******************************************************************************
* The function UART_direct_hex_put puts 4 bytes in hex directly (no ram buffer)
* to the UART.
*******************************************************************************/
void UART_direct_word_hex_put(uint32_t word)
{
  UART_direct_hex_put((word>>24)&0xFF);
  UART_TX_wait();
  UART_direct_hex_put((word>>16)&0xFF);
  UART_TX_wait();
  UART_direct_hex_put((word>>8)&0xFF);
  UART_TX_wait();
  UART_direct_hex_put(word&0xFF);
  UART_TX_wait();
}
/*******************************************************************************
* The function UART_direct_hex_int_put removes the leading zeroes and adds
  a decimal point. Best used in conjunction with hex2hexInt
*******************************************************************************/
void UART_direct_hex_int_put(uint32_t word, uint8_t deci)
{
  bool zeros = true;
  int8_t i = 7; //must be signed because of wrap-around
  for(i=7;i>=0;i--)
  {
    if(zeros && ((word>>(i*4))&0xF)==0);
    else
    {
      zeros = false;
      UART_low_nibble_direct_put((word>>i*4)&0xF);
      if(i == deci & deci>0) UART_direct_put('.');
    }
  }
}

```