Software FIFO Buffer for UART Communication

Background

UART communication is a very simple and inexpensive means of serial communication in embedded designs. Typically, a microcontroller has only a one or two-byte hardware buffer for each UART receiver and transmitter. If a multi-byte transmit is needed, the user needs to wait for the UART transmitter to shift out all of its data byte by byte so that data is not overwritten or discarded. This byte-by-byte "wait time" is amplified when a CPU is operating at higher speeds while using lower baud rates for UART communication. Software data buffers alleviate this problem by providing a larger buffer to store data while waiting for the UART to shift data out-of the transmitter or into the receiver.

Purpose

The purpose of this library is to provide a generic software buffer for UART communication, written in easy-to-understand C/C++ programming language. The buffer is interrupt driven which allows the microcontroller to execute functions in parallel with UART communication. Alternatively, the microcontroller can be put in a lower power mode (if supported) while waiting for UART TX or RX.

Prerequisites

- To use this software buffer, the microcontroller must support "UART TX buffer empty" and "UART RX data received" interrupts.
- It's assumed that the reader has some prior experience with C/C++ programming language.
- Having prior knowledge of FIFO buffers is beneficial, but not required. Here's some background info: link

Theory of Operation

The software buffer behaves like a circular First-In, First-Out (FIFO) buffer. Data is entered and removed from the buffer in chronological order. The size of the buffer is defined by FIFO_BUFFER_SIZE in *sw_fifo.h* and is limited by the amount of RAM in the microcontroller. The user should keep the size of the buffer as small as possible while still ensuring no overflow occurs.

The buffer uses external flags to indicate that data is present in the RX or TX software buffer. Any user-function can check the flags at any time to see if data has exists in either buffer. The flags are declared as external variables in *sw_fifo.h*. Therefore, in order to use the flags, the user must declare them in their own *main* file (or wherever the flag check is being made) as shown below:

```
volatile uint8_t uart_rx_fifo_not_empty_flag = 0; // this flag is
automatically set and cleared by the software buffer
volatile uint8_t uart_tx_fifo_not_empty_flag = 0; // this flag is
automatically set and cleared by the software buffer
```

The software buffer contains two flags for monitoring software buffer overflow conditions. These flags are automatically set by the software buffer when an overflow condition occurs. However, the user needs to manually clear the flags. If the buffer is made large enough, the user should be able get by without ever checking these flags. However, it is good practice to check if an overflow condition has occurred. These flags are declared as external variables in *sw_fifo.h*. Therefore, in order to use these flags, the user must declare the variables in their own *main* file (or wherever the flag check is being made) as shown below:

```
volatile uint8_t uart_rx_fifo_ovf_flag = 0;  // this flag is not
automatically cleared by the software buffer
volatile uint8_t uart_tx_fifo_ovf_flag = 0;  // this flag is not
automatically cleared by the software buffer
```

The software buffer also contains two flags for monitoring if each buffer is currently full. These flags are automatically handled by the software buffer and never need to be cleared by the user. Again, if the buffer is made large enough, the user should be able to get by without ever checking these flags. However, it is good practice to check if the buffer is full before writing to it to prevent overflow conditions. These flags are declared as external variables in *sw_fifo.h*. Therefore, in order to use these flags, the user must declare the variables in their own *main.c* file (or wherever the flag check is being made) as shown below:

```
volatile uint8_t uart_rx_fifo_full_flag = 0; // this flag is automatically
set and cleared by the software buffer
volatile uint8_t uart_tx_fifo_full_flag = 0; // this flag is automatically
set and cleared by the software buffer
```

To interact with this software buffer, the user's functions simply need to call the ***uart_send_byte()*** and ***uart_get_byte()*** functions. The user can monitor the 6 available software buffer flags if desired. For proper functionality, the user must enable global interrupts. The user also must enable the individual "TX hardware buffer empty" and "received valid data" interrupts specific to the microcontroller being used.

This software buffer works best on devices that have separate RX and TX hardware buffers. If the microcontroller uses the same hardware buffer for RX and TX, the "received data" interrupt will need to be disabled while data exists in the TX software buffer.

Customization

This software buffer was written in generic fashion. However, some portions are platform-specific (they will change depending on which microcontroller is being used). These areas are indicated by Double Comment Bars as shown below:

```
////////////////////////////////////
/* platform specific code required */
////////////////////////////////////
```

Error Handling

Receive Errors

- RX hardware buffer underflow errors should not occur because data is only added to the RX software buffer when the receive data interrupt executes.
- RX hardware buffer overflow is a possibility and should be handled by the user if desired.
- Frame and parity errors are possible and should be handled by the user if desired.
- RX software buffer overflow will never overwrite existing data. The "uart_rx_fifo_ovf_flag" is set when this event occurs and remains set until manually cleared by the user.
- RX software buffer underflow is possible. The ***uart_get_byte()*** function will return 0 under this condition. However, this can be avoided by checking the "uart_rx_fifo_not_empty_flag" before calling the ***uart_get_byte()*** function.

Transmit Errors

- TX hardware buffer overflow should not occur because the software buffer waits for the "TX buffer empty" interrupt to execute.
- TX hardware buffer underflow should not occur because the software buffer disables the "TX buffer empty" interrupt when the last element has been removed from the software buffer.
- TX software buffer overflow will never overwrite existing data. The "uart_tx_fifo_ovf_flag" is set when this event occurs and remains set until manually cleared by the user. A TX software buffer overflow condition can be avoided by checking the "uart_tx_fifo_full_flag" prior to calling the ***uart_send_byte()*** function.

C/C++ Code

sw_fifo.c

Download sw_fifo.c

```
////////////////////////////////////////////////////////////////////////
////////////////
/* enter necessary header files for proper interrupt vector and UART
/USART visibility */
////////////////////////////////////////////////////////////////////////
////////////////

#include <sw_fifo.h>

typedef struct {
  uint8_t  data_buf[FIFO_BUFFER_SIZE]; // FIFO buffer
  uint16_t i_first;                    // index of oldest data byte in
buffer
  uint16_t i_last;                     // index of newest data byte in
buffer
  uint16_t num_bytes;                  // number of bytes currently in
buffer
}sw_fifo_typedef;

sw_fifo_typedef rx_fifo = { {0}, 0, 0, 0 }; // declare a receive
software buffer
sw_fifo_typedef tx_fifo = { {0}, 0, 0, 0 }; // declare a transmit
software buffer


/***********************************************************************
****************************************/
// UART receive interrupt sub-routine
//  - interrupts when valid data exists in rx hardware buffer
//  - checks if there's room in the rx software buffer
//  - if there's room, it transfers the received data into the sw
buffer
//  - automatically handles "uart_rx_buffer_full_flag"
//  - sets overflow flag upon software buffer overflow (doesn't
overwrite existing data)
//////////////////////////////////////////////
/* enter name of UART RX IRQ Handler here */ {
//////////////////////////////////////////////

  /* Explicitly clear the source of interrupt if necessary */

  if(rx_fifo.num_bytes == FIFO_BUFFER_SIZE) {      // if the sw buffer
is full
```

```
    uart_rx_fifo_ovf_flag = 1;                     // set the overflow
flag
  }else if(rx_fifo.num_bytes < FIFO_BUFFER_SIZE) { // if there's room
in the sw buffer

    ///////////////////////////////////////////////////
    /* read error/status reg here if desired         */
    /* handle any hardware RX errors here if desired */
    ///////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
/////////////
    rx_fifo.data_buf[rx_fifo.i_last] = /* enter pointer to UART rx
hardware buffer here */ // store the received data as the newest data
element in the sw buffer

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
/////////////

    rx_fifo.i_last++;                              // increment the
index of the most recently added element
    rx_fifo.num_bytes++;                           // increment the
bytes counter
  }
  if(rx_fifo.num_bytes == FIFO_BUFFER_SIZE) {      // if sw buffer just
filled up
    uart_rx_fifo_full_flag = 1;                    // set the RX FIFO
full flag
  }
  if(rx_fifo.i_last == FIFO_BUFFER_SIZE) {         // if the index has
reached the end of the buffer,
    rx_fifo.i_last = 0;                            // roll over the
index counter
  }
  uart_rx_fifo_not_empty_flag = 1;                 // set received-data
flag
} // end UART RX IRQ handler
/***********************************************************************
****************************************/


/***********************************************************************
****************************************/
// UART transmit interrupt sub-routine
//  - interrupts when the tx hardware buffer is empty
//  - checks if data exists in the tx software buffer
//  - if data exists, it places the oldest element of the sw buffer
into the tx hardware buffer
//  - if the sw buffer is emptied, it disables the "hw buffer empty"
interrupt
```

```
//  - automatically handles "uart_tx_buffer_full_flag"
//////////////////////////////////////////////
/* enter name of UART TX IRQ Handler here */ {
//////////////////////////////////////////////

  /* Explicitly clear the source of interrupt if necessary */

  if(tx_fifo.num_bytes == FIFO_BUFFER_SIZE) { // if the sw buffer is
full
    uart_tx_fifo_full_flag = 0;               // clear the buffer full
flag because we are about to make room
  }
  if(tx_fifo.num_bytes > 0) {                 // if data exists in the
sw buffer


///////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
    /* enter pointer to UART tx hardware buffer here */ = tx_fifo.
data_buf[tx_fifo.i_first]; // place oldest data element in the TX
hardware buffer

///////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////

    tx_fifo.i_first++;                        // increment the index of
the oldest element
    tx_fifo.num_bytes--;                      // decrement the bytes
counter
  }
  if(tx_fifo.i_first == FIFO_BUFFER_SIZE) {   // if the index has
reached the end of the buffer,
    tx_fifo.i_first = 0;                      // roll over the index
counter
  }
  if(tx_fifo.num_bytes == 0) {                // if no more data exists

    uart_tx_fifo_not_empty_flag = 0;          // clear flag


///////////////////////////////////////////////////////////////////////
//
    /* disable UART "TX hw buffer empty" interrupt
here                     */
    /* if using shared RX/TX hardware buffer, enable RX data interrupt
here */

///////////////////////////////////////////////////////////////////////
//

  }
}// end UART TX IRQ handler
/***********************************************************************
```

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


/************************************************************************
****************************************/
// UART data transmit function
//  - checks if there's room in the transmit sw buffer
//  - if there's room, it transfers data byte to sw buffer
//  - automatically handles "uart_tx_buffer_full_flag"
//  - sets the overflow flag upon software buffer overflow (doesn't
overwrite existing data)
//  - if this is the first data byte in the buffer, it enables the "hw
buffer empty" interrupt
void uart_send_byte(uint8_t byte) {

  ///////////////////////////////////////////////////////////
  /* disable interrupts while manipulating buffer pointers */
  ///////////////////////////////////////////////////////////

  if(tx_fifo.num_bytes == FIFO_BUFFER_SIZE) {      // no room in the sw
buffer
    uart_tx_fifo_ovf_flag = 1;                     // set the overflow
flag
  }else if(tx_fifo.num_bytes < FIFO_BUFFER_SIZE) { // if there's room
in the sw buffer
    tx_fifo.data_buf[tx_fifo.i_last] = byte;       // transfer data
byte to sw buffer
    tx_fifo.i_last++;                              // increment the
index of the most recently added element
    tx_fifo.num_bytes++;                           // increment the
bytes counter
  }
  if(tx_fifo.num_bytes == FIFO_BUFFER_SIZE) {      // if sw buffer is
full
    uart_tx_fifo_full_flag = 1;                    // set the TX FIFO
full flag
  }
  if(tx_fifo.i_last == FIFO_BUFFER_SIZE) {         // if the "new data"
index has reached the end of the buffer,
    tx_fifo.i_last = 0;                            // roll over the
index counter
  }

  ///////////////////////
  /* enable interrupts */
  ///////////////////////

  if(tx_fifo.num_bytes > 0) {                      // if there is data
in the buffer

        uart_tx_fifo_not_empty_flag = 1;               // set flag
////////////////////////////////////////////////////////////////////////
///
    /* if using shared RX/TX hardware buffer, disable RX data interrupt
here */
    /* enable UART "TX hw buffer empty" interrupt
here                       */

////////////////////////////////////////////////////////////////////////
///

  }
}
/***********************************************************************
****************************************/


/***********************************************************************
****************************************/
// UART data receive function
//  - checks if data exists in the receive sw buffer
//  - if data exists, it returns the oldest element contained in the
buffer
//  - automatically handles "uart_rx_buffer_full_flag"
//  - if no data exists, it clears the uart_rx_flag
uint8_t uart_get_byte(void) {

  ///////////////////////////////////////////////////////////
  /* disable interrupts while manipulating buffer pointers */
  ///////////////////////////////////////////////////////////

  uint8_t byte = 0;
  if(rx_fifo.num_bytes == FIFO_BUFFER_SIZE) { // if the sw buffer is
full
    uart_rx_fifo_full_flag = 0;               // clear the buffer full
flag because we are about to make room
  }
  if(rx_fifo.num_bytes > 0) {                 // if data exists in the
sw buffer
    byte = rx_fifo.data_buf[rx_fifo.i_first]; // grab the oldest
element in the buffer
    rx_fifo.i_first++;                        // increment the index of
the oldest element
    rx_fifo.num_bytes--;                      // decrement the bytes
counter
  }else{                                      // RX sw buffer is empty
    uart_rx_fifo_not_empty_flag = 0;          // clear the rx flag
  }
  if(rx_fifo.i_first == FIFO_BUFFER_SIZE) {   // if the index has
reached the end of the buffer,
    rx_fifo.i_first = 0;                      // roll over the index
counter
  }
  ///////////////////////
  /* enable interrupts */
  ///////////////////////

  return byte;                                // return the data byte
}
/************************************************************************
*****************************************/
```

sw_fifo.h

Download sw_fifo.h

```
#define FIFO_BUFFER_SIZE 128 // software buffer size (in bytes)

// UART data transmit function
//  - checks if there's room in the transmit sw buffer
//  - if there's room, it transfers data byte to sw buffer
//  - automatically handles "uart_tx_buffer_full_flag"
//  - sets the overflow flag upon software buffer overflow (doesn't
overwrite existing data)
//  - if this is the first data byte in the buffer, it enables the "hw
buffer empty" interrupt
void uart_send_byte(uint8_t byte);


// UART data receive function
//  - checks if data exists in the receive sw buffer
//  - if data exists, it returns the oldest element contained in the
buffer
//  - automatically handles "uart_rx_buffer_full_flag"
//  - if no data exists, it clears the uart_rx_flag
uint8_t uart_get_byte(void);

volatile extern uint8_t uart_rx_fifo_not_empty_flag; // this flag is
automatically set and cleared by the software buffer
volatile extern uint8_t uart_rx_fifo_full_flag;      // this flag is
automatically set and cleared by the software buffer
volatile extern uint8_t uart_rx_fifo_ovf_flag;       // this flag is
not automatically cleared by the software buffer
volatile extern uint8_t uart_tx_fifo_full_flag;      // this flag is
automatically set and cleared by the software buffer
volatile extern uint8_t uart_tx_fifo_ovf_flag;       // this flag is
not automatically cleared by the software buffer
volatile extern uint8_t uart_tx_fifo_not_empty_flag; // this flag is
automatically set and cleared by the software buffer
```

Example Usage

The following code shows how the user could interact with the UART software buffer from the *main* function:

```
#include <sw_fifo.h> // make software buffer visible to this file

volatile uint8_t uart_rx_fifo_not_empty_flag = 0;
volatile uint8_t uart_rx_fifo_full_flag      = 0;
volatile uint8_t uart_rx_fifo_ovf_flag       = 0;
volatile uint8_t uart_tx_fifo_full_flag      = 0;
volatile uint8_t uart_tx_fifo_ovf_flag       = 0;
volatile uint8_t uart_tx_fifo_not_empty_flag = 0;

int main (void) {
   uint8_t i = 0;
   uint8_t rx_data = 0;

   // initialize clocks
   // disable global interrupts
   // initialize gpio
   // initialize uart/usart

   // enable "UART RX" interrupt and "TX hardware buffer empty"
interrupt
   // enable global interrupts

   while(uart_tx_fifo_full_flag);    // wait for room to open up in the
software buffer
   uart_send_byte('A');              // transmit ASCII character 'A'
   while(uart_tx_fifo_full_flag);
   uart_send_byte(0x41);             // transmit ASCII character 'A'

   // transmit ASCII characters 1-5
   for(i=0; i<5; i++) {
      while(uart_tx_fifo_full_flag);
      uart_send_byte(i+48);
   }

   while(1) {
      // enter sleep mode if supported (wake from UART Rx)

      while(uart_rx_fifo_not_empty_flag) {     // if data exists in
software buffer
         rx_data = uart_get_byte();            // grab first data byte
from software buffer

         /* handle received byte as desired */

         uart_send_byte(rx_data);              // example of how to
echo received ASCII characters
      }

      // check for rx overflow condition
      if(uart_rx_fifo_ovf_flag) {
            /* handle rx overflow condition as desired */

            uart_rx_fifo_ovf_flag = 0;  // clear the rx overflow flag
         }

         // check for tx overflow condition
         if(uart_tx_fifo_ovf_flag) {

            /* handle tx overflow condition as desired */

            uart_tx_fifo_ovf_flag = 0;  // clear the tx overflow flag
         }

         // if you need to disable global interrupts, you should wait
   until the tx fifo is empty
         while(uart_tx_fifo_not_empty_flag);
         /* Disable global interrupts */

         /* Do something */

         /* Re-enable global interrupts */
      } // end while
   } // end main
```

Contact the Author

The nice thing about software buffers is that you can tailor them to your application. You can make them as large or small as you want and by using an interrupt driven software buffer, you can perform other functions in parallel with UART communication. If your microcontroller supports it, you can enter a lower power mode while waiting for incoming characters over UART. I implemented this software buffer on a microcontroller that supported "wake from UART RX interrupt" sleep mode and it worked like a champ! The goal of this page was to provide a generic C/C++ based software buffer that can be applied to any microcontroller. I hope you find it useful and easy to understand. I have successfully implemented it on a few different microcontrollers. The code can be used freely, however it is a "use at your own risk" scenario. If you have questions, feedback, or would like to see something added to the eewiki, let me know at eewiki@digikey.com. Happy coding!

- Scott