How to use the AVR USART to implement serial links

The USART (or UART) has bidirectional serial data transfer. It is almost impossible to use this efficiently without interrupts. Here are some guidelines, to help you understand the sample code.

- Check the clock frequency & that the baud rate required can be achieved. Error of <1% is never a problem, <2% will work but is undesirable, 5% will not work. Note that this assumes that the other end of the link has perfect timing, which will normally be true for a PC or USB serial port. If the other end of the link is another microcontroller check its exact baud rate and try to match that. Check the serial link configuration which must match the other end - most common is 8 bit, no parity, 1 stop bit.
- If possible use RTS/CTS handshake lines so that flow control is automatic. Otherwise characters may be lost due to over-runs. If you use software flow control (xon/xoff) these two ASCII characters cannot be used as data. Thus binary data transmission is not possible without encoding.
- Check your hardware connections. Microprocessors & PCs will have TXD serial output & RXD serial input. Devices that expect to be connected to computers (Zigbit modules) will have input TXD & output RXD. Normally therefore you connect TXD to TXD, RXD to RXD. However to connect a microprocess to a PC you must have RXD -> TXD, TXD->RXD. Similarly RTS/CTS must be connected either right way round or reversed. Always check datasheets.
- Use an interrupt handler to process received data. If you use synchronous (non-interrupt) transmission then the application will be unable to do anything else when data is being transmitted. If this is a problem use an interrupt handler also for transmitted data.
- How to send & receive characters:
  - Use Rx complete interrupt/status to determine when to load next received byte. This interrupt (or status bit) is automatically cleared by reading the USART data buffer.
  - Use UDR empty interrupt/status to determine when to output next transmitted byte. This interrupt or status bit is automatically cleared by writing to the USART status buffer.
  - Don't use Tx complete interrupt/status since it determines when last byte has actually finished transmission. Normally it is not useful to know this.
- **Transmission using interrupts requires care** because if no characters are available to transmit the ISR must return without loading the USART data buffer, and therefore switch off the interrupt. When a new character becomes available the interrupt will not happen again, and no characters are transmitted. The solution is to switch interrupts on again whenever a character is transmitted. The sample code can be used as a template since it solves this problem correctly.
- **Reception using interrupts is easier.** Often character processing may actually be done within the interrupt handler function. Otherwise, if a buffer is used, the interrupt handler

adds characters to a memory buffer and a non-interrupt receive function (as below) extracts them. See sample code.

- The buffers used in memory for transmitted and/or received characters should for complete generality be circular buffers implemented using an array with indexes for first and last stored character which wrap around when the reach the end. Note that it is necessary to store the number of characters in the buffer as well to distinguish between the full & empty cases.
- A simpler solution is available when the length of transmitted or received messages is known and when the transmission (or reception) of messages need not overlap their generation/processing. In that case a non-circular memory buffer can be used to hold a single message.

Sample code for USART

See the next page for some sample code (given with working project in avrcode.zip) which implements UART Rx & Tx via interrupts. CPU used here is AtMega88p (updated low voltage Atmega8) code will port to any CPU with USART - but may need to change names of registers (e.g. UCSR0B->UCSRB). This code uses both Rx & Tx interrupts. Baud rate 38400, 8 data bits, no parity, 1 stop bit. Transmit characters are stored in a circular queue. Receive characters are handled by user-defined routine **process_rx_char()** which is called from the receive interrupt and must not be too long. If long actions are needed in response to received data these should be implemented in non-interrupt code which polls a global memory location set from interrupt code, as below:

```
void process_rx_char( char ch)
{
   static char rxbuffer[RXBUFFERSIZE];
   static uint8_t  rxBufferIndex=0;
   rxBuffer[rxBufferIndex++] = ch; /* put next char in buffer */
   if <received message is complete> { /* detemined from length of content */
      if (NextMessageReady == TRUE) { /* over-run error */ };
      /* transfer data from rxBuffer[] to MessageBuffer[] */
      rxBufferIndex = 0; /* get ready for next character
      NextMessageReady = TRUE; /* signal non-interrupt code */
   }
}

/* non-interrupt code */
static char MessageBuffer[RXBUFFERSIZE];
static uint8_t NextMessageReady;

{
   /* Initialise */
   NextMessageReady = FALSE;
   /* loop processing received messages */
   for (;;) {
        while (NextMessageReady = FALSE); /* wait for next message */
        /* process next message in MessageBuffer */
        NextMessageReady = FALSE; /* reset handshake variable */
        /* do other stuff */
}
```

Sample Code (available online in avrcode.zip)

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <macros.h>
#include <uart.h>



#define QUEUE_FULL_ERROR 1
#define OK 0
#define TX_BUFF_SIZE 20
#define Q_INCR(x) x = x+1; if (x == TX_BUFF_SIZE) x = 0;
#define UDRIE_on() UCSR0B = 0x98 | BIT(UDRIE0)
#define UDRIE_off() UCSR0B = 0x98

#define NCTS (PORTB & 0x04)

/* data structures for queue */
struct q_t {
unsigned char head_pos;
unsigned char empty_pos;
unsigned char length;
char buffer[TX_BUFF_SIZE];
};

static struct q_t txq;

//initialise q structures
static void usart_q_init(void)
{
        txq.head_pos=0;
        txq.empty_pos=0;
        txq.length=0;
}

//USART initialize
// desired baud rate: 38400
// actual: baud rate:38462 (0.2%)
void usart_init(void)
{
 UCSR0B = 0x00; //disable while setting baud rate
 UCSR0A = 0x02;
 UCSR0C = 0x06;
 UBRR0L = 0x0C; //set baud rate lo
 UBRR0H = 0x00; //set baud rate hi
 UCSR0B = 0x98;

 EIMSK = 0x01; //enable ext interrupt 0
 EIFR = 0x02; // falling edge
 usart_q_init();
}

//send char ch using interrupts
//on exit interrupts will be enabled
//regardless of state on entry
unsigned char usart_send(char ch)
{
        unsigned char len;
        CLI();//disable interrupts
        if ( (BIT(UDRE0) & UCSR0A) && !NCTS) {
                // buffer is empty so send now
                UDR0 = ch;
                return OK;
        }
        //otherwise check queue full
        if ((len=txq.length) == TX_BUFF_SIZE) {
                //return with error if full
                SEI(); //enable interrupts
                return QUEUE_FULL_ERROR;
        }
        //add ch to buffer
        txq.buffer[txq.empty_pos] = ch;
        Q_INCR(txq.empty_pos);
        txq.length = len+1;
        if (!NCTS) UDRIE_on(); //switch INT on
        SEI();//enable interrupts
        return OK;
}

//USART RX interrupt handler

ISR(USART_RX_vect){
        //uart has received a character in UDR
        // process the received char right away
  process_rx_char(UDR0);
}

//USART TX interrupt handler
ISR(USART_UDRE_vect)
{
        //character transferred to shift register
        //so UDR is now empty
        //try to get char from tx buffer
        if (txq.length != 0) {
                //get char and send it
                UDR0 = txq.buffer[txq.head_pos];
                Q_INCR(txq.head_pos);
                txq.length = txq.length-1;
        } else {
                UDRIE_off(); // switch off
interrupt since no data
        }
}

// INT0 Handler
ISR(INT0_vect)
{
  if (NCTS) {
        UDRIE_off();
```

```
          EIFR &= ~0x01;
          }
  else {
          UDRIE_on();
          EIFR |= 0x01;
          }
}
```