AVR306: Using the AVR® UART in C

Features
- Setup and Use of the AVR UART
- Code Examples for Polled and Interrupt Controlled UART
- Compact Code
- C Code Included for AT90S8515

Description
This application note describes how to set up and use the UART present in most AVR devices. C code examples are included for polled and interrupt controlled UART applications.

Polled UART
The application is continuously checking the UDRE bit in the UART Status Register to control when the UART has finished sending a byte. When receiving data, the application is continuously checking the RXC bit in the UART Status Register to control when the UART has completed receiving a byte.

Interrupt Controlled UART
The UART generates an interrupt when the UART has finished transmitting or receiving a byte. The interrupt handling routines uses modulo 2^n addressing of circular buffers for buffering incoming and outgoing data. The buffer sizes must be defined before using the routines. Set the UART_RX_BUFFER_SIZE and UART_TX_BUFFER_SIZE variables to the buffer size in bytes. Note that these variables must be a power of 2. If not, a compiler error message will be flagged.

An extra function is added to the UART2 example code. The DataInReceiveBuffer returns zero if the receive buffer does not contain any data. This function does, in contrast to the ReceiveByte function, not wait for incoming data, but returns immediately the status of the buffer. Note: this routine does not return the number of bytes in the buffer.

Table 1.

<table>
<thead>
<tr>
<th>Polled UART</th>
<th>Interrupt controlled UART</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compact code</td>
<td>Reasonable code size</td>
</tr>
<tr>
<td>Application busy while communicating</td>
<td>Application free while communicating</td>
</tr>
</tbody>
</table>
Usage

Both examples use the same set of routines. If other devices than AT90S8515 is used, the include file in the code must be changed accordingly.

void InitUART( unsigned char baudrate );

Enables the UART and sets the baud rate. Using baud rates that differs more than ±0.5% is not recommended. Please refer to the UART section in the data sheet for selecting the baud rate. The value passed to this function will be written to the UART Baud Rate Register.

unsigned char ReceiveByte( void );

Waits for one byte to be received and returns it’s value.

void TransmitByte( unsigned char data );

Waits for transmission to be allowed, sends byte given as parameter to the UART transmitter and returns.

unsigned char DataInReceiveBuffer( void );

Returns zero (0) if the receive buffer is empty.

UART1.c - Polled

/* includes */
#include <io8515.h>

/* UART Control Register Bit Definitions */
#define RXCIE 7
#define TXCIE 6
#define UDRIE 5
#define RXEN 4
#define TXEN 3
#define CHR9 2
#define RXB8 1
#define TXB8 0

/* UART Status Register Bit Definitions */
#define RXC 7
#define TXC 6
#define UDRE 5
#define FE 4
#define OVR 3

/* Prototypes */
void InitUART( unsigned char baudrate );
unsigned char ReceiveByte( void );
void TransmitByte( unsigned char data );

/* main - a simple test program*/
void main( void )
{
    InitUART( 11 ); /* set the baudrate to 19,200 bps using a 3.6864MHz crystal */
    while ( 1 )
    {
        TransmitByte( ReceiveByte() ); /* echo the received character */
    }
}
/* initialize UART */
void InitUART( unsigned char baudrate )
{
    UBR = baudrate;  /* set the baud rate */
    UCR = ( (1<<RXEN) | (1<<TXEN) );  /* enable UART receiver and transmitter */
}

/* Read and write functions */
unsigned char ReceiveByte( void )
{
    while ( !(USR & (1<<RXC)) )  /* wait for incoming data */
    ;  /* return the data */
    return UDR;
}

void TransmitByte( unsigned char data )
{
    while ( !(USR & (1<<UDRE)) )  /* wait for empty transmit buffer */
    ;  /* start transmission */
    UDR = data;
}

UART2.c - Interrupt Driver

/* includes */
#include <io8515.h>
#include <ina90.h>

/* UART Control Register Bit Definitions */
#define RXCIE 7
#define TXCIE 6
#define UDRIE 5
#define RXEN 4
#define TXEN 3
#define CHR9 2
#define RXB8 1
#define TXB8 0

/* UART Status Register Bit Definitions */
#define RXC 7
#define TXC 6
#define UDRE 5
#define FE 4
#define OVR 3

/* UART Buffer Defines */
#define UART_RX_BUFFER_SIZE 128 /* 1,2,4,8,16,32,64,128 or 256 bytes */
#define UART_RX_BUFFER_MASK ( UART_RX_BUFFER_SIZE - 1 )
#if ( UART_RX_BUFFER_SIZE & UART_RX_BUFFER_MASK )
#error RX buffer size is not a power of 2
#endif

/* Static Variables */
static unsigned char UART_RxBuf[UART_RX_BUFFER_SIZE];
static volatile unsigned char UART_RxHead;
static volatile unsigned char UART_RxTail;

static unsigned char UART_TxBuf[UART_TX_BUFFER_SIZE];
static volatile unsigned char UART_TxHead;
static volatile unsigned char UART_TxTail;

/* Prototypes */
void InitUART( unsigned char baudrate );
unsigned char ReceiveByte( void );
void TransmitByte( unsigned char data );

main - a simple test program*/
void main( void )
{
    InitUART( 11 ); /* set the baudrate to 19,200 bps using a 3.6864MHz crystal */
    _SEI(); /* enable interrupts => enable UART interrupts */

    while ( 1 ) /* forever */
    {
        TransmitByte( ReceiveByte() ); /* echo the received character */
    }
}

/* initialize UART */
void InitUART( unsigned char baudrate )
{
    unsigned char x;

    UBRR = baudrate; /* set the baud rate */
    /* enable UART receiver and transmitter, and receive interrupt */

    UCR = ( (1<<RXCIE) | (1<<RXEN) | (1<<TXEN) );

    x = 0; /* flush receive buffer */
AVR306

UART_RxTail = x;
UART_RxHead = x;
UART_TxTail = x;
UART_TxHead = x;
}

/* interrupt handlers */
interrupt [UART_RX_vect] void UART_RX_interrupt( void )
{
    unsigned char data;
    unsigned char tmphead;

    data = UDR; /* read the received data */
    /* calculate buffer index */
    tmphead = ( UART_RxHead + 1 ) & UART_RX_BUFFER_MASK;
    UART_RxHead = tmphead; /* store new index */

    if ( tmphead == UART_RxTail )
    {
        /* ERROR! Receive buffer overflow */
    }
    UART_RxBuf[tmphead] = data; /* store received data in buffer */
}

interrupt [UART_UDRE_vect] void UART_TX_interrupt( void )
{
    unsigned char tmptail;
    /* check if all data is transmitted */

    if ( UART_TxHead != UART_TxTail )
    {
        /* calculate buffer index */
        tmptail = ( UART_TxTail + 1 ) & UART_TX_BUFFER_MASK;
        UART_TxTail = tmptail; /* store new index */

        UDR = UART_TxBuf[tmptail]; /* start transmission */
    }
    else
    {
        UCR &= ~(1<<UDRIE); /* disable UDRE interrupt */
    }
}

/* Read and write functions */
unsigned char ReceiveByte( void )
{
    unsigned char tmptail;
    while ( UART_RxHead == UART_RxTail ) /* wait for incoming data */
tmptail = ( UART_RxTail + 1 ) & UART_RX_BUFFER_MASK; /* calculate buffer index */

UART_RxTail = tmptail; /* store new index */

return UART_RxBuf[tmptail]; /* return data */

void TransmitByte( unsigned char data )
{
    unsigned char tmphead; /* calculate buffer index */

    tmphead = ( UART_TxHead + 1 ) & UART_TX_BUFFER_MASK; /* wait for free space in buffer */
    while ( tmphead == UART_TxTail );

    UART_TxBuf[tmphead] = data; /* store data in buffer */
    UART_TxHead = tmphead; /* store new index */

    UCR |= (1<<UDRIE); /* enable UDRE interrupt */
}

unsigned char DataInReceiveBuffer( void )
{
    return ( UART_RxHead != UART_RxTail ); /* return 0 (FALSE) if the receive buffer is empty */
}