

Embedded Systems Programming:

LAB2

In the first lab we got ourselves familiar with the required software tools, I/O ports, external interrupt handling and timers. In this lab we will use Universal Synchronous Asynchronous Receiver Transmitter (USART) to carry out serial communication to an external device. Objective is to understand the functioning of USART and get it done on AVR atmega32 microcontroller. First go through this handout and code given below.

Normally microcontroller based solution requires interfacing of microcontroller with sensors, devices or computer. In many occasions, microcontroller requires to make this interfacing using serial or parallel communication with external devices or computer. In this lab we will learn about one of this communication interface, USART and how to achieve it between *atmega32* and PC (using *minicom* terminal emulation program). First we will go through basics of USART, then required parameter calculation and then run sample program to achieve it with atmega32.

USART is one of the important communication protocols for microcontroller. It takes data bytes and transmits it bit by bit in sequential manner. Following figure 1 is the minimum pin diagram for USART (RS-232) where pin2 ,3 are data transmitting and receiving pins while pin5 is ground pin.

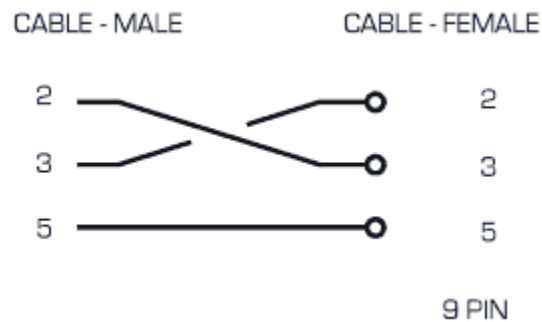


Figure 1: Partial pin diagram for 9 pin RS232 connector

Atmega32 can work with four different data transfer modes for USART namely, normal asynchronous, double speed asynchronous, master synchronous, slave synchronous. From these most common mode of operation is normal asynchronous mode. In Atmega32 the important parameter to configure for USART are baud rate, number of data bits, parity and number of stop bits. Here baud rate defines at what rate this devices communicate with each other and other parameter defines the frame structure for the communication which is shown below in figure 2.

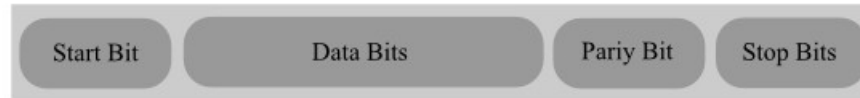


Figure 2: Frame format for USART

Baud Rate Register (UBRR) is used where value is stored for down counter. Each time the down-counter value reaches zero a clock is generated. You can calculate what UBRR value you need to write depending on desired baud rate and MCU clock speed (1MHz) by formula:

$$BAUD = \frac{f_{clk}}{16(UBRR + 1)} \quad (1)$$

Using formula (1), you can calculate the required baud rate. Next you need to configure the control and status register for USART to set required frame format (Number of data bits, Parity bit and Stop bits).

On PC side you need to run *minicom* program which is terminal emulation program for Linux. You can run it by writing *minicom* at shell prompt. In that you have to *configure minicom* and *comm. parameters*.

serialCom.c

Explanation:

It's echo program which receives data on USART and transmits it back to USART.

Library Files:

- avr/io.h
- avr/interrupt.h
- avr/signal.h
- inttypes.h

Library Functions:

- *sei*

Functions:

- *port_init*
- *global_init*
- *usart_init*
- *gvar_init*
- *init*
- *usart_transmit*
- *usart_receive*
- *SIGNAL*

- *main*

Registers:

- GICR
- MCUCSR
- MCUCR
- UBRRH-UBRRL
- UCSR-A,B,C
- UDRB
- UDRA
- PORTB

Local Variables:

- unsigned char data

Global Variables:

- uint8_t led, state

Program 2.1:

```
#include<inttypes.h>
#include<avr/interrupt.h>
#include<avr/signal.h>
#include<avr/io.h>

void          usart_transmit(unsigned char data);
unsigned char usart_receive(void);

SIGNAL(SIG_UART_RECV)
{
    unsigned char data;
    data = UDR;
    usart_transmit(data);
    PORTB=~PORTB;
}

SIGNAL(SIG_UART_TRANS)
{
    //    PORTB=~PORTB;
}

void port_init()
{
    DDRB      = 0xFF;
    DDRA      = 0x00;
}
```

```

void global_init()
{
    GICR          = (1<<INT0) | (1<<INT1);
    MCUCSR        = MCUCSR | 0x80;
    MCUCSR        = MCUCSR | 0x80;
    MCUCR         = (1<<ISC01) | (1<< ISC11);
    MCUCR         = (1<<ISC01) | (1<< ISC11);
}

void usart_init()
{
    UBRRH         = 0x00 ;
    UBRRL         = 0x19;
    UCSRA         = (0x00); //(1<<RXC); //interrupt
    UCSRB         = (1<<RXEN) | (1<<TXEN) | (1<<RXCIE) | (1<<TXCIE);
                //interrupt
    UCSRC         = (1<<URSEL) | (1<<USBS) | (3<<UCSZ0);
                // 8bit, 2stop and no parity
    SREG          = (1 << 7);
}

unsigned char usart_receive(void)
{
    unsigned char data;
    while ( !(UCSRA & (1<<RXC)) );
    data =UDR;
    return data;
}

void usart_transmit(unsigned char data)
{
    while ( !( UCSRA & (1<<UDRE)) );
    UDR = data;
}

void gvar_init()
{
    PORTB         = 0x00;
}

init()
{
    port_init();
    global_init();
    usart_init();
    gvar_init();
}

int main(void)
{
    init();
    sei();
    for(;;)
    {
    }
}

```

Circuit-Kit Setup:

To connect circuit-kit for above experiment we need 4 cables.

- (1) Serial cable to connect serial port of computer to programmable serial port of kit,
- (2) 8-wire cable to connect port-b to LEDs of kit
- (3) Cable to select target microcontroller slot to program.
- (4) 2-wire cable to connect port-D (D0, D1) to stk-500 RXD-TXD.
- (5) You may need to use external crystal if you find errors in communication.

Compile the program and download it on the microcontroller on the kit. Change the connection of RS-232 cable from STK-500 RS-232 control port to, STK-500 RS-232 spare port.

Expected Output:

On proper cable connection, program upload and *minicom* configuration you will see echo of characters you write on *minicom* terminal.

Your task:

In your handout you have been given a program which echoes the character on *minicom*. In this program what ever you write on *minicom* program will send it back on *minicom* to redisplay. Using this write a program to displays repeatedly *a text* banner (e.g. “This is ATmega32”) on *minicom*.

Now, normally banner has on time and off time, so that it appears only in on times and in off times it does not show up. Design proper mechanism to incorporate such effect in your banner. To do this, take on-off time from *minicom*. Further many of you like to implement the other setting for the banner such speed of text and emergency stop and reset.