MODULE -3 : INTERFACING MICROCONTROLLER WITH EXTERNAL PERIPHERALS

RS232 Standards

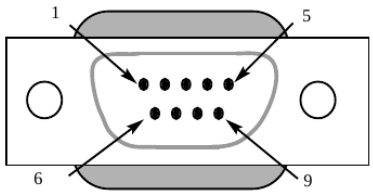
RS232 is an interfacing standard introduced by Electronics Industries Association (EIA). It is one of the most widely used serial I/O interfacing standards. RS232 standards allows data communication among equipments of various manufactures. In RS232, a 1 is represented by −3 to −25 V, while a 0 bit is +3 to +25 volts, making −3 to +3 undefined. For this reason, to connect any RS232 to a microcontroller system we must use voltage converters such as MAX232 to convert the voltage levels. MAX232 IC chips are commonly referred to as line drivers.

In data communication, equipments are classified as:

- Data Terminal Equipment (DTE). Eg: Computers
- Data Communication Equipment (DCE). Eg: Modems

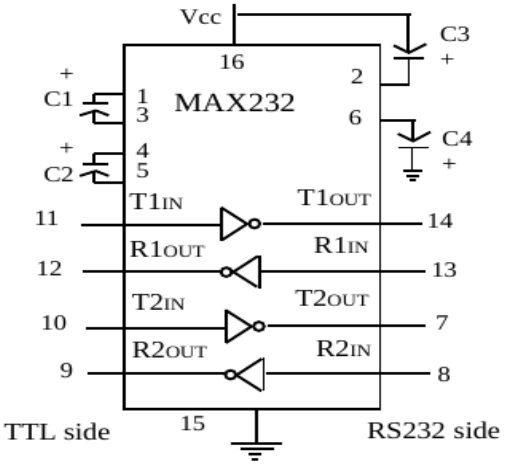
ATMEGA32 CONNECTION TO RS232

In ATmega32, the two pins TX (pin 15) and RX (pin 14) are used for transferring and receiving data serially. These are part of PortD. These pins are TTL compatible. So we need line drivers such as MAX232 to convert voltage levels. Following figure shows 9 pin RS232 Connector. In this connector, Pin 2 is RX (Receive) pin, Pin 3 is TX (Transmit) pin and Pin 5 is ground.



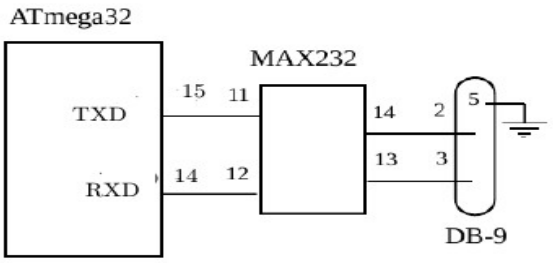
MAX232

- The MAX232 converts from RS232 voltage levels to TTL voltage levels, and vice versa.
- The MAX232 chip uses a +5 V power source. AVR microcontroller also uses +5V power source. So by using a single 5V power supply, we can power both the AVR and MAX232
- The MAX232 has two sets of line drivers for transferring and receiving data (T1, T2 and R1, R2).
- Following figure shows the pin details of MAX232



- MAX232 requires four capacitors. To save board space, some designers use the MAX233 chip, it eliminates the need of capacitors.
- The MAX233 chip is much more expensive than the MAX232 and not pin compatible

Figure shows the connection diagram of Atmega32 with RS232 (DB9).



40-Pin DIP Package ATmega32

AVR Serial Port Programming:

The AVR microcontroller has a built-in USART (universal synchronous asynchronous receiver-transmitter), which is a commonly used IC for serial data communication. Following five registers are used for AVR serial port programming:

- UDR (USART Data Register)
  - It is the buffer register to hold the data to be transmitted or received serially
- UCSRA, UCSRB, UCSRC (USART Control Status Register)
  - Used to control the serial data transfer
- UBRR (USART Baud Rate Register)
  - Used to set the baud rate (data transfer speed)

Programming Steps to transfer data serially (Asynchronous mode)

1. Enable USART transmitter by loading USCRB register with the value 08H
2. The UCSRC register is loaded with the value 06H, indicating asynchronous mode with 8-bit data frame, no parity, and one stop bit.
3. The UBRR is loaded with the value for the desired baud rate
4. The character byte to be transmitted serially, is written into the UDR register.
5. Monitor the UDRE bit of the UCSRA register to make sure UDR is ready for the next byte.
6. To transmit the next character, go to Step 4.

Programming Steps to receive data serially (Asynchronous mode)

1. Enable USART receiver by loading USCRB register with the value 10H
2. The UCSRC register is loaded with the value 06H, indicating asynchronous mode with 8-bit data frame, no parity, and one stop bit.
3. The UBRR is loaded for the desired baud rate.
4. The RXC flag bit of the UCSRA register is monitored for a HIGH to see if an entire character has been received yet.
5. When RXC is raised, the UDR register contains the received byte. Its contents are moved into a safe place.
6. To receive the next character, go to Step 5.

Example 1: Write a C function to initialize the USART to work at 9600 baud, 8-bit data, and 1 stop bit. Assume XTAL = 8 MHz.

Solution:

```
void usart_init (void)
{
  UCSRB = 0x08;    //enable transmitter
  UCSRC = 0x06;    //Asynchronous, 8 bit data, no parity, one stop bit
  UBRRL = 0x33;    // the hex value for 9600 baud
}
```

Example 2: Write a C program for the AVR to transfer the letter 'G' serially at 9600 baud, continuously. Use 8-bit data and 1 stop bit. Assume XTAL = 8 MHz.

Solution:

For this program, we have to monitor the UDRE flag in UCSRA register. This flag is set when the transmit data buffer (UDR) is empty and it is ready to receive new data

```
void usart_init (void)
{
  UCSRB = 0x08;    //enable transmitter
  UCSRC = 0x06;    //Asynchronous, 8 bit data, no parity, one stop bit
  UBRRL = 0x33;    // the hex value for 9600 baud
}

void usart_send (unsigned char ch)
{
  while (! (UCSRA & (1<<UDRE))); // wait until UDR is empty
  UDR = ch;                      // transmit the letter
}

int main (void)
{
  usart_init();
  while(1)
    usart_send ('G');
  return 0;
}
```

Example 3: Modify the example 2 to send the message "The Earth is but One Country. " to the serial port continuously.

Solution:

For this program, the main function can be modified as:

```
int main (void)
{
  unsigned char str[30]= "The Earth is but One Country. ";
  unsigned char strLength = 30;
  unsigned char i = 0;
  usart_init();
  while(1)
  {
    usart_send(str[i++]);
    if (i >= strLength)
      i = 0;
  }
  return 0;
}
```

Example 4: Write a program in AVR C to receive bytes of data serially and put them on Port A. Set the baud rate at 9600, 8-bit data, and 1 stop bit.

Solution:

For this program, we have to monitor the RXC flag of UCSRA register. This flag will set when the received data is ready in the buffer.

```
#include <avr/io.h>
int main (void)
{
  DDRA = 0xFF;   //Port A is input
  UCSRB = 0x10;  //Enable receiver
  UCSRC = 0x06;  //Asynchronous, 8 bit data, no parity, one stop bit
  UBRRL = 0x33;
  while(1)
  {
    while (! (UCSRA & (1<<RXC)));  //wait until new data
    PORTA = UDR;
  }
  return 0;
}
```