

Embedded Systems

Dr. Abdelhay Ali

Lecture 7

USART

Outlines

1. Introduction
2. Serial data framing
3. Wires and Hardware connection
4. Programming of USART in AVR
5. Programming steps
6. Example
7. Programming AVR Serial communication using Interrupt

Outlines

1. Introduction
2. Serial data framing
3. Wires and Hardware connection
4. Programming of USART in AVR
5. Programming steps
6. Example
7. Programming AVR Serial communication using Interrupt

Introduction

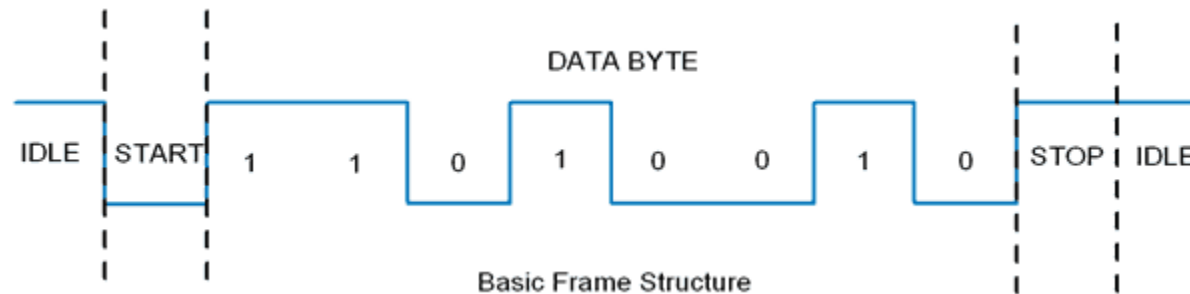
- ❑ **A universal synchronous and asynchronous receiver-transmitter (USART)** is a type of a serial interface device that can be programmed to communicate asynchronously or synchronously
- ❑ AVR ATmega has flexible USART, which can be used for serial communication with other devices like computers, serial GSM, GPS modules, etc.

Outlines

1. Introduction
2. Serial data framing
3. Wires and Hardware connection
4. Programming of USART in AVR
5. Programming steps
6. Example
7. Programming AVR Serial communication using Interrupt

Serial data framing

- While sending/receiving data, some bits are added for the purpose of knowing the beginning/ending of data, etc. commonly used structure is: 8 data bits, 1 start bit (logic 0), and 1 stop bit (logic 1), as shown



There are also other supported frame formats available in UART, like parity bit, variable data bits (5-9 data bits).

Serial data framing

Speed (Baud rate)

As we know the bit rate is “Number of bits per second (bps)”, also known as Baud rate in Binary system. Normally this defines how fast the serial line is. There are some standard baud rates defined e.g. 1200, 2400, 4800, 19200, 115200 bps, etc. Normally 9600 bps is used where speed is not a critical issue.

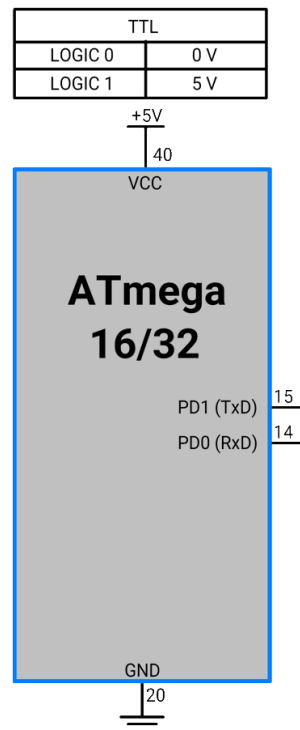
Outlines

1. Introduction
2. Serial data framing
3. Wires and Hardware connection
4. Programming of USART in AVR
5. Programming steps
6. Example
7. Programming AVR Serial communication using Interrupt

Wires and Hardware connection

Normally in USART, we only need Tx (Transmit), Rx(Receive), and GND wires.

- AVR ATmega USART has a TTL voltage level which is 0 v for logic 0 and 5 v for logic 1.



Wires and Hardware connection

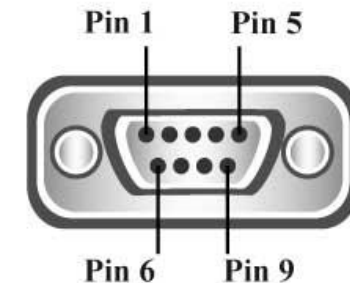
- In computers and most of the old devices, RS232 protocol is used for serial communication, where normally 9 pin 'D' shape connector is used. RS232 serial communication has different voltage levels than ATmega serial communication i.e. +3 v to +25 v for logic zero and -3 v to -25 v for logic 1.

RS232



Pin 1	DCD
Pin 2	RXD
Pin 3	TXD
Pin 4	DTR
Pin 5	GND
Pin 6	DSR
Pin 7	RTS
Pin 8	CTS
Pin 9	RI

RS232 Pinout (9 Pin Male)



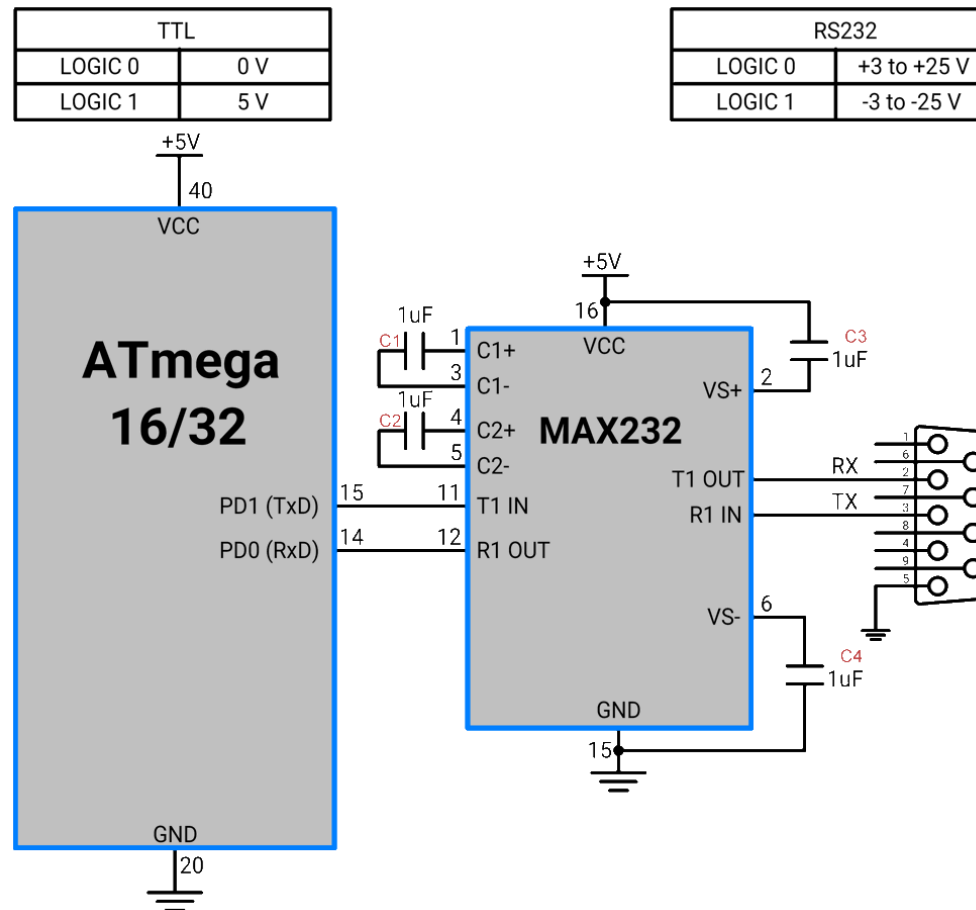
Although there are 9 pins in the DB9 connector, we don't need to use all the pins. Only 2nd Tx(Transmit), 3rd Rx(Receive), and 5th GND pin need to be connected.

Wires and Hardware connection

- So to communicate with RS232 protocol, we need to use a voltage level converter like MAX232 IC.



Wires and Hardware connection



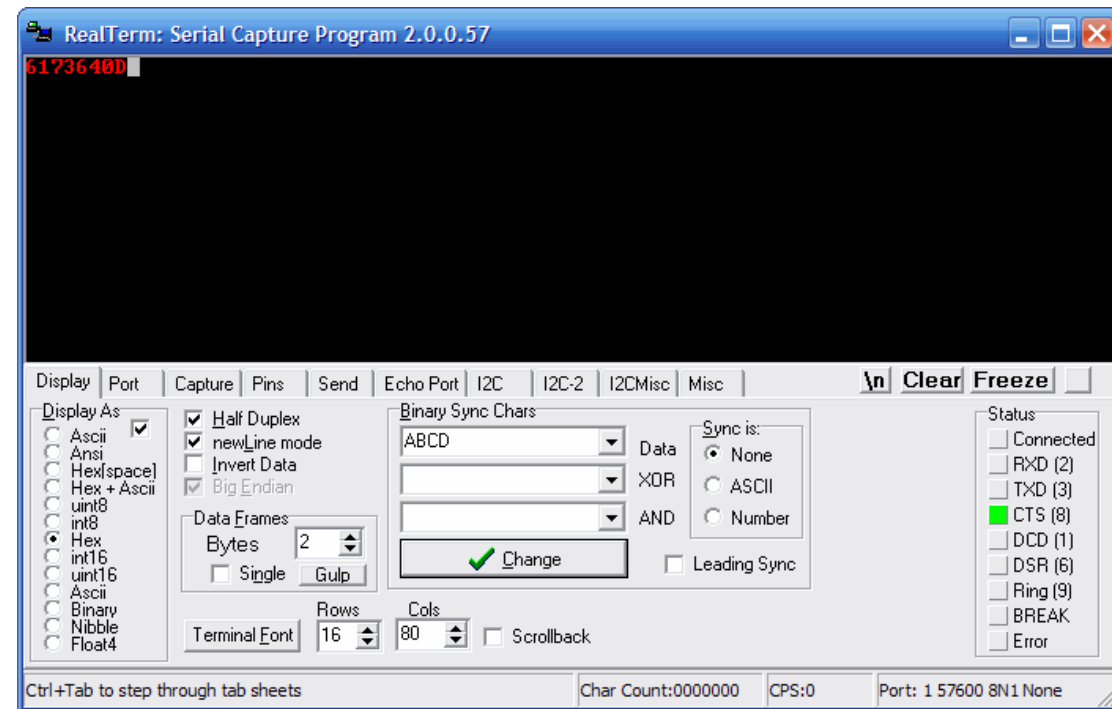
Wires and Hardware connection

With a new PC and laptops, there is no RS232 protocol and DB9 connector. We have to use serial to the USB connector. There are various serial to USB connectors available e.g. CP2102, FT232RL, CH340, etc.



Wires and Hardware connection

To see serial communication, we can use a serial terminal like Realterm, Teraterm, etc. By selecting the serial port number (COM port in windows) and baud rate, we can open a serial port for communication.



Wires and Hardware connection

As stated in the datasheet ATmega 16 USART has the following features

- ✓ Full Duplex Operation (Independent Serial Receive and Transmit Registers)
- ✓ Asynchronous or Synchronous Operation
- ✓ Master or Slave Clocked Synchronous Operation
- ✓ High-Resolution Baud Rate Generator
- ✓ Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits
- ✓ Odd or Even Parity Generation and Parity Check Supported by Hardware
- ✓ Data OverRun Detection
- ✓ Framing Error Detection
- ✓ Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter
- ✓ Three Separate Interrupts on TX Complete, TX Data Register Empty, and RX Complete
- ✓ Multi-processor Communication Mode
- ✓ Double Speed Asynchronous Communication Mode

Outlines

1. Introduction
2. Serial data framing
3. Wires and Hardware connection
- 4. Programming of USART in AVR**
5. Programming steps
6. Example
7. Programming AVR Serial communication using Interrupt

Programming of USART in AVR

AVR basic Registers

1. **UDR: USART Data Register**

It has basically two registers, one is Tx. Byte and the other is Rx Byte. Both share the same UDR register. Do remember that, when we write to the UDR reg. Tx buffer will get written and when we read from this register, Rx Buffer will get read. Buffer uses the FIFO shift register to transmit the data.

2. **UCSRA**: USART Control and Status Register A. As the name suggests, is used for control and status flags. In a similar fashion, there are two more USART control and status registers, namely UCSRB and UCSRC.

3. **UBRR**: USART Baud Rate Register, this is a 16-bit register used for the setting baud rate.

Programming of USART in AVR

UCSRA: USART Control and Status Register A

7	6	5	4	3	2	1	0
RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

- **Bit 7 – RXC:** USART Receive Complete

This flag bit is set when there is unread data in UDR. The RXC Flag can be used to generate a Receive Complete interrupt.

- **Bit 6 – TXC:** USART Transmit Complete

This flag bit is set when the entire frame from Tx Buffer is shifted out and there is no new data currently present in the transmit buffer (UDR). The TXC Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXC Flag can generate a Transmit Complete interrupt.

Programming of USART in AVR

UCSRA: USART Control and Status Register A

7	6	5	4	3	2	1	0
RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

- **Bit 5 – UDRE:** USART Data Register Empty

If UDRE is one, the buffer is empty which indicates the transmit buffer (UDR) is ready to receive new data. The UDRE Flag can generate a Data Register Empty Interrupt. UDRE is set after a reset to indicate that the transmitter is ready.

- **Bit 4 – FE:** Frame Error

- **Bit 3 – DOR:** Data OverRun

This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters) and a new character is waiting in the receive Shift Register.

Programming of USART in AVR

UCSRA: USART Control and Status Register A

7	6	5	4	3	2	1	0
RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

- **Bit 2 – PE:** Parity Error
- **Bit 1 – U2X:** Double the USART Transmission Speed
- **Bit 0 – MPCM:** Multi-processor Communication Mode

Programming of USART in AVR

UCSRB: USART Control and Status Register B

7	6	5	4	3	2	1	0
RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

- **Bit 2 – PE:** Parity Error
- **Bit 1 – U2X:** Double the USART Transmission Speed
- **Bit 0 – MPCM:** Multi-processor Communication Mode

Programming of USART in AVR

UCSRB: USART Control and Status Register B

7	6	5	4	3	2	1	0
RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

- **Bit 7 – RXCIE:** RX Complete Interrupt Enable
Writing one to this bit enables interrupt on the RXC Flag.
- **Bit 6 – TXCIE:** TX Complete Interrupt Enable
Writing one to this bit enables interrupt on the TXC Flag.
- **Bit 5 – UDRIE:** USART Data Register Empty Interrupt Enable
Writing one to this bit enables interrupt on the UDRE Flag.

Programming of USART in AVR

UCSRB: USART Control and Status Register B

7	6	5	4	3	2	1	0
RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

- **Bit 4 – RXEN:** Receiver Enable

Writing one to this bit enables the USART Receiver.

- **Bit 3 – TXEN:** Transmitter Enable

Writing one to this bit enables the USART Transmitter.

- **Bit 2 – UCSZ2:** Character Size

The UCSZ2 bits combined with the UCSZ1:0 bit in UCSRC sets the number of data bits (Character Size) in a frame the receiver and transmitter use.

- **Bit 1 – RXB8:** Receive Data Bit 8

- **Bit 0 – TXB8:** Transmit Data Bit 8

Programming of USART in AVR

UCSRC: USART Control and Status Register C

7	6	5	4	3	2	1	0
URSEL	UMSEL	UPM1	UPM0	USBS	USCZ1	USCZ0	UCPOL

- **Bit 7 – URSEL:** Register Select

This bit selects between accessing the **UCSRC** or the **UBRRH** Register, as both register shares the same address. The URSEL must be one when writing the UCSR or else data will be written in the UBRRH register.

- **Bit 6 – UMSEL:** USART Mode Select

This bit selects between the Asynchronous and Synchronous mode of operation.

0 = Asynchronous Operation

1 = Synchronous Operation

Programming of USART in AVR

UCSRC: USART Control and Status Register C

7	6	5	4	3	2	1	0
URSEL	UMSEL	UPM1	UPM0	USBS	USCZ1	USCZ0	UCPOL

- **Bit 5:4 – UPM1:0: Parity Mode**

These bits enable and set the type of parity generation and check. If parity a mismatch is detected, the PE Flag in UCSRA will be set.

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Programming of USART in AVR

UCSRC: USART Control and Status Register C

7	6	5	4	3	2	1	0
URSEL	UMSEL	UPM1	UPM0	USBS	USCZ1	USCZ0	UCPOL

- **Bit 3 – USBS:** Stop Bit Select

This bit selects the number of Stop Bits to be inserted by the Transmitter. The Receiver ignores this setting.

0 = 1-bit

1 = 2-bit

Programming of USART in AVR

UCSRC: USART Control and Status Register C

7	6	5	4	3	2	1	0
URSEL	UMSEL	UPM1	UPM0	USBS	USCZ1	USCZ0	UCPOL

•Bit 2:1 – UCSZ1:0: Character Size

The UCSZ1:0 bits combined with the UCSZ2 bit in UCSRB sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Programming of USART in AVR

UCSRC: USART Control and Status Register C

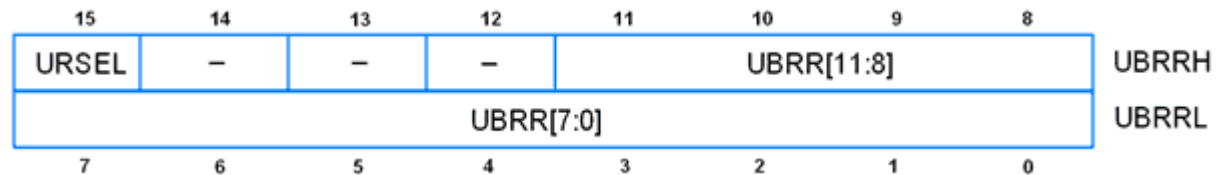
7	6	5	4	3	2	1	0
URSEL	UMSEL	UPM1	UPM0	USBS	USCZ1	USCZ0	UCPOL

- **Bit 0 – UCPOL:** Clock Polarity

This bit is used for synchronous mode only. Write this bit to zero when the asynchronous mode is used

Programming of USART in AVR

UBRRL and UBRRH: USART Baud Rate Registers

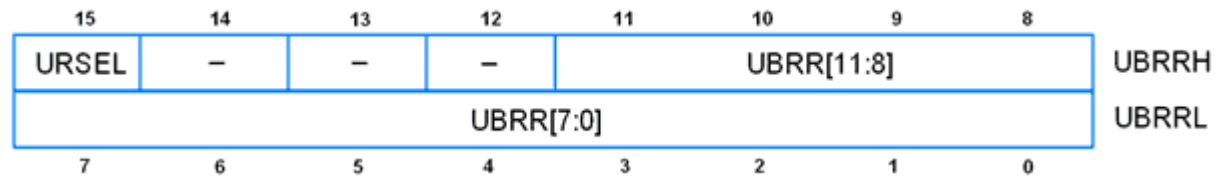


- Bit 15 – URSEL:** Register Select

This bit selects between accessing the **UCSRC** or the **UBRRH** Register, as both register shares the same address. The URSEL must be one when writing the UCSRC or else data will be written in the UBRRH register.

Programming of USART in AVR

UBRRL and UBRRH: USART Baud Rate Registers



- **Bit 11:0 – UBRR11:0:** USART Baud Rate Register.
Used to define the baud rate

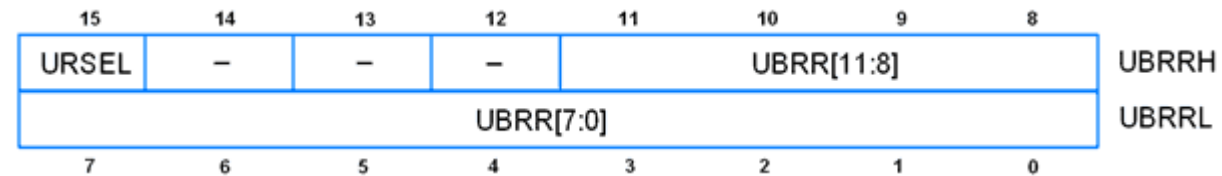
$$UBRR = \frac{F_{osc}}{16 * BaudRate} - 1$$

$$BaudRate = \frac{F_{osc}}{16 * (UBRR + 1)}$$

Example: suppose F_{osc} =8 MHz and required baud rate= 9600 bps.
Then the value of **UBRR= 51.088 i.e. 51.**

Programming of USART in AVR

UBRRL and UBRRH: USART Baud Rate Registers



- **Bit 11:0 – UBRR11:0:** USART Baud Rate Register.
Used to define the baud rate

$$UBBR = \frac{F_{osc}}{16 * BaudRate} - 1$$

$$BaudRate = \frac{F_{osc}}{16 * (UBBR + 1)}$$

```
#define F_CPU 8000000UL          /* Define frequency here its 8MHz */
#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)
```


Outlines

1. Introduction
2. Serial data framing
3. Wires and Hardware connection
4. Programming of USART in AVR
5. **Programming steps**
6. Example
7. Programming AVR Serial communication using Interrupt

Programming steps

Initialization of USART

1. Enable transmission and reception by using the UCSRB register.
2. Set data bit size to 8 bit by using the UCSRC register.
3. Set baud rate using the UBRR register.

```
void UART_init(long USART_BAUDRATE)
{
    UCSRB |= (1 << RXEN) | (1 << TXEN); /* Turn on transmission and reception */
    UCSRC |= (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1); /* Use 8-bit char size */
    UBRRL = BAUD_PRESCALE; /* Load lower 8-bits of the baud rate */
    UBRRH = (BAUD_PRESCALE >> 8); /* Load upper 8-bits */
}
```

Programming steps

Receiving Character

Monitor the RXC bit in UCSRA. RXC bit indicates receive complete status.

```
unsigned char UART_RxChar()
{
    while ((UCSRA & (1 << RXC)) == 0); /* Wait till data is received */
    return(UDR); /* Return the byte */
}
```

Programming steps

Transmitting Character

Monitor the UDRE bit from UCSRA. When the UDRE flag becomes one, which indicates the transmitting buffer is empty and ready to accept another byte.

```
void UART_TxChar(char ch)
{
    while (! (UCSRA & (1<<UDRE))); /* Wait for empty transmit buffer */
    UDR = ch ;
}
```

Programming steps

Transmitting Character

Monitor the UDRE bit from UCSRA. When the UDRE flag becomes one, which indicates the transmitting buffer is empty and ready to accept another byte.

```
void UART_SendString(char *str)
{
    unsigned char j=0;
    while (str[j]!=0) /* Send string till null */
    {
        UART_TxChar(str[j]);
        j++;
    }
}
```

Outlines

1. Introduction
2. Serial data framing
3. Wires and Hardware connection
4. Programming of USART in AVR
5. Programming steps
6. **Example**
7. Programming AVR Serial communication using Interrupt

Example

Program for echo test. In this program, the ATmega 16 controller will receive a character and send it back. We can use the PC terminal to observe echo.

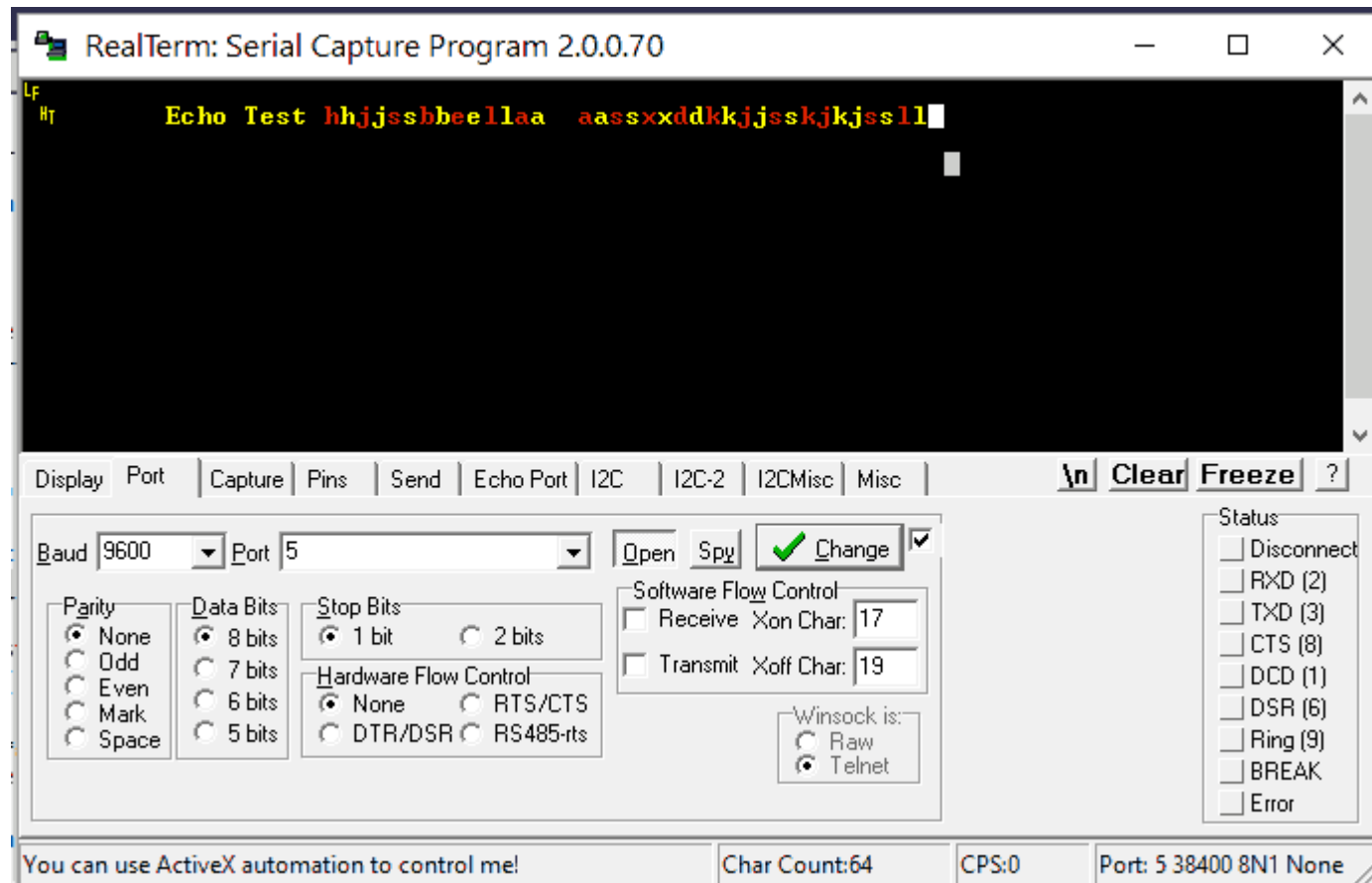
```

#define F_CPU 8000000UL /* Define frequency here its 8MHz */
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>
#include <stdio.h>
//#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

void UART_init(long USART_BAUDRATE);
unsigned char UART_RxChar();
void UART_TxChar(char ch);
void UART_SendString(char *str);

void main(void)
{
    char c;
    UART_init(9600);
    UART_SendString("\n\t Echo Test ");
    while(1) {
        c=UART_RxChar();
        UART_TxChar(c);
    }
}

```

Outlines

1. Introduction
2. Serial data framing
3. Wires and Hardware connection
4. Programming of USART in AVR
5. Programming steps
6. Example
7. Programming AVR Serial communication using Interrupt

Programming AVR Serial communication using Interrupt

Up till now, we have seen serial programming using the polling method, which wastes the time of the controller.

To enable interrupt-based transmission we have to set HIGH the USART Data Register Empty Interrupt Enable (UDRIE).

To enable interrupt-based reception we need to set HIGH the Receive Complete Interrupt Enable(RXCIE) bit in UCSRB.

```

#define F_CPU 8000000UL /* Define frequency here its 8MHz */
#include <avr/io.h>
#include <avr/interrupt.h>
#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

void UART_init(long USART_BAUDRATE) {
    UCSRB |= (1 << RXEN) | (1 << RXCIE); /* Turn on the transmission and reception */
    UCSRC |= (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1); /* Use 8-bit character sizes */
    UBRRL = BAUD_PRESCALE; /* Load lower 8-bits of the baud rate */
    UBRRH = (BAUD_PRESCALE >> 8); /* Load upper 8-bits */
}

ISR(USART_RXC_vect) {
    PORTB=UDR;
}

int main()
{
    UART_init(9600);
    sei(); DDRB=0xFF;
    while(1);
}

```

Thanks

