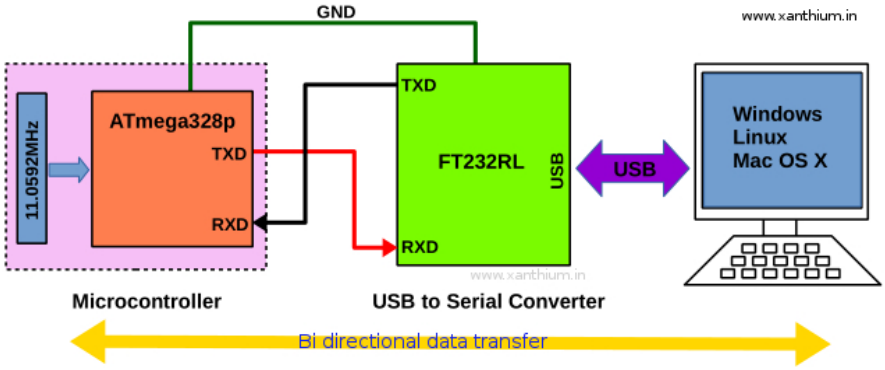
ATmega328P to PC Serial Communication using USART Tutorial

Submitted by RahulSreedharan on 9 July 2019 - 9:26am



In this tutorial we will learn how to program the **USART(uart) of ATmega328P microcontroller to communicate with a Linux/ Windows PC** using asynchronous serial communication protocol.

The ATmega328p microcontroller will send and receive data (ASCII Strings) to a PC running either a Linux or Windows operating system using it's (ATmega328p) UART pins.

The data will be received/transmitted by a terminal emulation program like **PuTTY** or Tera Term.

if you are looking for **ATmega RS485 Communication** check this tutorial.

Sourcecodes

- The code for the microcontroller section is written in **embedded C** and compiled using **AVR-GCC (WinAVR)** .
- Hex code is uploaded into Atmega328p with **AVRDUDE** using **USBasp** Programmer.
- All Source codes including "Make" files are available on our Github Repo.

Hardware Connection Setup

We are using a 32 pin TQFP version of the ATmega328p microcontroller .If you are using the 28 pin DIP version,please make sure that the pin numbers match by referring to the data sheet of ATmega328p.

- ATmega328p Datasheet

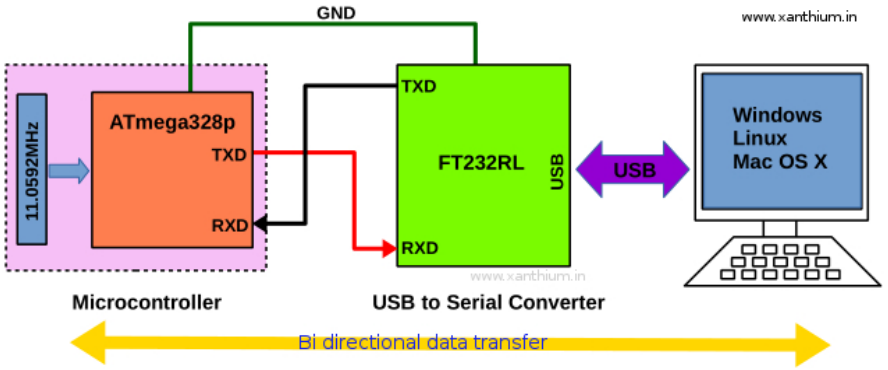
ATmega328p is clocked with an external **Quartz crystal running at 11.0592MHz**.

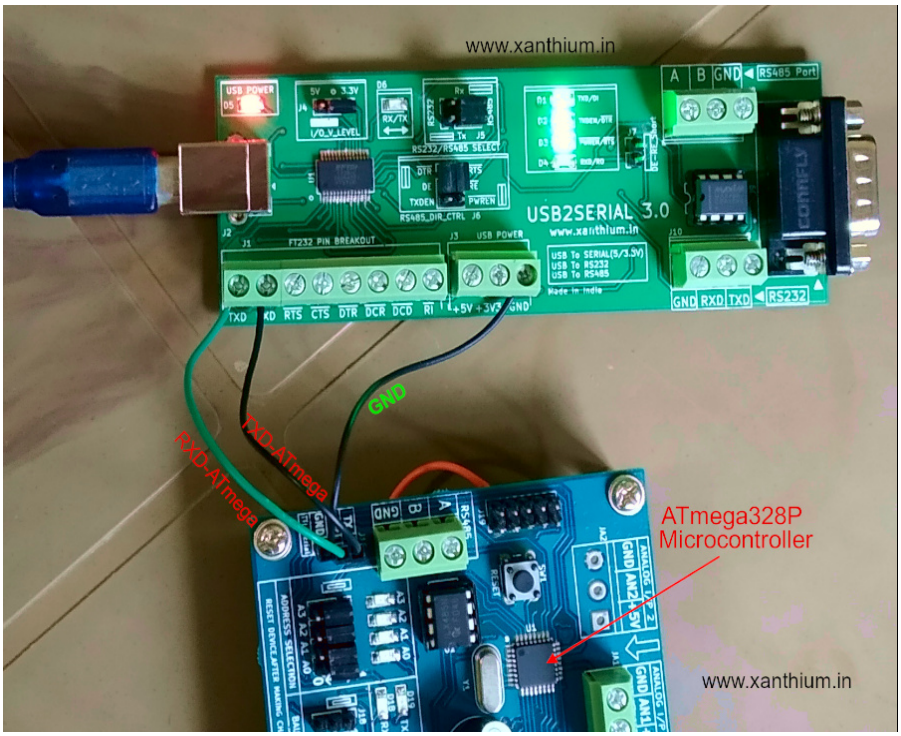
You can check this tutorial to learn how to configure ATmega328 to use an external crystal as clock source.

Please note that ,

- **TXD** of ATmega328 is connected to **RXD** of USB to Serial Converter
- RXD of ATmega328 is connected to TXD of USB to Serial Converter

The following block diagram shows the hardware connections of the setup.





Since the PC cannot receive the UART signals directly from the ATmega328P microcontroller,a USB to Serial Converter based on FT232RL is used to convert the serial TTL signals to USB Protocol.

Buy it Online

ATmega328p USART

Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a configurable peripheral of ATmega328p which supports both Synchronous (SPI) and asynchronous(Serial) communication protocols.

In our case we will be dealing only with asynchronous communication.

USART Initialization

The USART has to be initialized before any communication can take place.The initialization process consists of the following steps.

- Setting the Baud rate
- Setting the frame format (here 8N1)
- Enabling the Transmitter or Receiver depending upon the usage.

Setting the Baud rate

USART supports all commonly used **baud rates from 2400 bps to 230.4kps without any issues**.You can go up to 2.5Mbps for a crystal frequency of 20MHz.The maximum we are able to achieve on a Windows7 machine is 230.4k bps .

The required baudrate can be selected by writing the corresponding value to the UBBRn register.

The ATmega328p datasheet has a section on commonly used Baud rates,Error rates,Crystal Frequencies and their corresponding UBBRn values.You can look up that table on the datasheet and select your desired UBBRn values.

Table 24-4. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies

Baud Rate [bps]	f_{osc} = 1.0000MHz				f_{osc} = 1.8432MHz				f_{osc} = 2.0000MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max.(1)	62.5kbps		125kbps		115.2kbps		230.4kbps		125kbps		250kbps	

UBBRn values under U2Xn =1 column are for double the USART transmission speed which you can ignore.

If you check the table in the datasheet,you can find that **error rates are close to zero when you are using fractional clock rates** (f_{OSC}) like 1.84MHz,3.68MHz,7.37MHz or 11.0592MHz.

In our case we are using a crystal frequency of 11.0592MHz,the table below shows the corresponding UBBRn values,Error rates for a f_{osc} value of 11.0592MHz.

Table 24-6. Examples of UBRRn Settings for Commonly

Baud Rate [bps]	f_{osc} = 11.0592MHz			
	U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error
2400	287	0.0%		
4800	143	0.0%		
9600	71	0.0%		
14.4k	47	0.0%		
19.2k	35	0.0%		
28.8k	23	0.0%		
38.4k	17	0.0%		
57.6k	11	0.0%		
76.8k	8	0.0%		
115.2k	5	0.0%		
230.4k	2	0.0%		

Setting the Frame rate.

A serial frame is defined to be a stream of data bits with synchronization bits (start and stop bits),and optionally a parity bit for error checking.

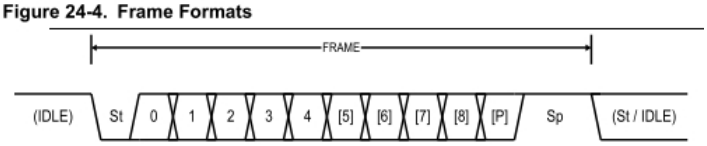
USART support the following combination of start/Stop/data/parity bits.

- 1 Start bit
- 5, 6, 7, **8**, or 9 Data bits
- No,EVEN or ODD parity bit
- 1 or 2 stop bits

A frame starts with the start bit,
followed by the data bits (from five up to nine data bits in total): first the least significant data bit, then the next data bits ending with the most significant bit.
If enabled, the parity bit is inserted after the data bits, before the one or two stop bits.

When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state.

Figure 24-4. Frame Formats



In our case we are using the 8N1 setup ie

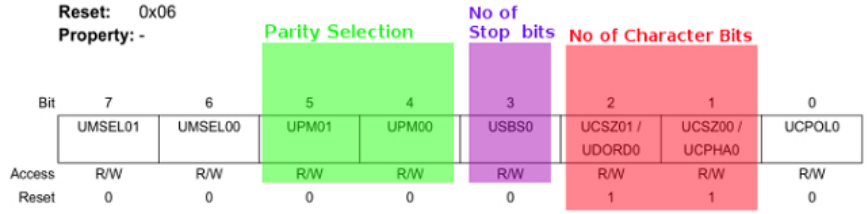
- 1 Start Bit

- **8** Data Bits
- **N**o Parity
- **1** Stop Bit

USART Control and Status Register 0 C or UCSR0C controls frame settings of the atmega328 USART.

24.12.4. USART Control and Status Register 0 C

Name: UCSR0C
Offset: 0xC2
Reset: 0x06
Property: -



Since we are using the 8N1 format , we don't have to change anything as the default settings (0x06) gives 8N1 configuration.

Setting up as Transmitter or Receiver

24.12.3. USART Control and Status Register 0 B

Name: UCSR0B
Offset: 0xC1
Reset: 0x00
Property: -

USART Control and Status Register 0 B or **UCSR0B** contains the bits for configuring the USART as Transmitter,Receiver or Both.

- Enabling **TXEN0** bit configures USART as **Transmitter**
- Enabling **RXEN0** bit configures USART as **Receiver**

Configuring either of the bits(TXEN0 or RXEN0) overrides normal port functions of the corresponding pins ie GPIO pins are either configured as TXD or RXD.

ATmega328 Transmitting a String to PC

We are going to continuously transmit an ASCII string "Hello from ATmega328p" to the PC.

PuTTY program running on PC will receive it and display it on console.

The transmission speed is set at **230.4 kbps** in **8N1** format.

The code shows how the program is organized for transmitting data ASCII string "Hello from ATmega328p" to the PC,

Please note that

- Code below is partial and may be missing some parts.
- **Please use the full code from our github repository**.

```
#define BAUD_RATE_230400_BPS  2 // 230.4kps

int main()
{
    int i = 0;
    unsigned int ubrr = BAUD_RATE_230400_BPS;
    unsigned char data[] = "Hello from ATmega328p  ";

    /* Set Baudrate  */
    UBRR0H = (ubrr>>8);
    UBRR0L = (ubrr);

    UCSR0C = 0x06;       /* Set frame format: 8data, 1stop bit  */
    UCSR0B = (1<<TXEN0); /* Enable  transmitter                 */

    while(1) /* Loop the messsage continously */
    {
        i = 0;
        while(data[i] != 0) /* print the String  "Hello from ATmega328p" */
        {
          while (!( UCSR0A & (1<<UDRE0))); /* Wait for empty transmit buffer*/
          UDR0 = data[i];            /* Put data into buffer, sends the data */
          i++;                             /* increment counter           */
        }
      }
}//end of main
```

The UBBRn value for a Baudrate of 230.4 kbps is found to be 2 from the ATmega datasheet and is stored in the 16 bit variable(unsigned int) `ubrr` in the above program.

The UBBRn register is a 12 bit register consisting of 2 registers

- USART Baud Rate 0 Register Low **UBRR0L** (8bit)
- USART Baud Rate 0 Register High **UBRR0H** (4 bits are used)

```
/* Set Baudrate  */
    UBRR0H = (ubrr>>8);
    UBRR0L = (ubrr);
```

- Shift the 16bit value `ubrr` 8 times to the right and **transfer the upper 8 bits to UBBR0H** register.
- Copy the 16 bit value `ubrr` to the 8 bit UBBR0L register,Upper 8 bits are truncated while **lower 8 bits are copied**.

After the ubbr value is transferred into the two registers (UBBR0L and UBBR0H) ,We set the frame rate and enable the transmitter of ATmega328p.

```
UCSR0C = 0x06;       /* Set frame format: 8data, 1stop bit  */
UCSR0B = (1<<TXEN0); /* Enable  transmitter                 */
```

UCSR0C = 0x06 is rebundant as it is the default value for the register.

Writing a byte to the **UDR0 register of ATmega328p USART will intiate data transmission**.

After the transmission **UDRE0 flag is set to 1**,indicating that buffer is ready to transmit again.

UDRE0 bit is found in the **UCSR0A** register.

24.12.2. USART Control and Status Register 0 A

Name: UCSR0A
Offset: 0xC0
Reset: 0x20
Property: -

Bit	7	6	5	4	3	2	1	0
	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0
Access	R	R/W	R	R	R	R	R/W	R/W
Reset	0	0	1	0	0	0	0	0

We then use a `while()` loop to wait till the data has been transmitted by the USART.

```
while (!( UCSR0A & (1<<UDRE0)));  /* Wait for empty transmit buffer*/
```

When USART is transmitting a byte (Buffer is full) ,UDRE0 = 0 ,

So (UCSR0A & (1<<UDRE0)) = 0 and

!(UCSR0A & (1<<UDRE0)) = 1

which means **while loop is true ie while(1)** ,So execution flow stays at the while loop.

When USART has finished transmitting a byte (buffer is empty) ,UDRE0 = 1 ,

So (UCSR0A & (1<<UDRE0)) = 1 and

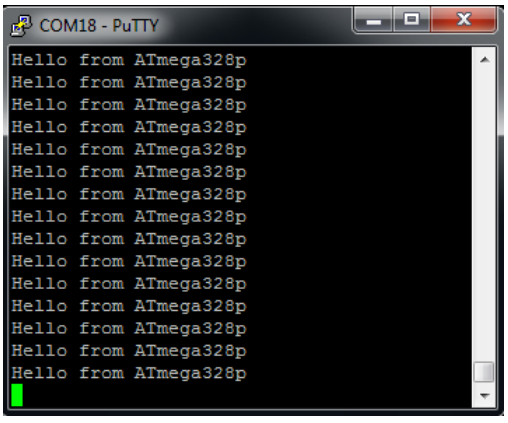
!(UCSR0A & (1<<UDRE0)) = 0

which means **while loop is false ie while(0)** ,So execution flow exits the loop and goes down .

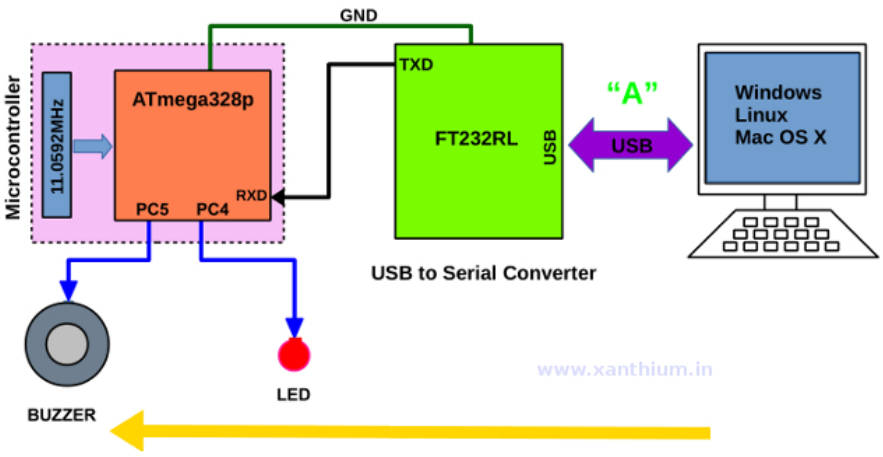
We transfer each character from the array to UDR0 to be transmitted byte by byte into the PC .

After the contents of the array are transmitted completely, `'\n' '\r'` characters are send (refer github code) so that the string is formatted perfectly on the PuTTY console Window.

The Below image shows the received string in a PuTTY Console Window.



ATmega328 Receiving a Character from PC



Here we are going to receive an ASCII character from PC through the serial port as shown in the above picture.The transmission baud rate is 230.4k bps.

- An LED is connected to **PC4** which will blink when the ATmega328P receives the character **'A'** from PC
- A buzzer is connected to **PC5** which will beep when the ATmega328P receives the character **'B'** from PC

Characters are send through the USB Virtual serial port of the PC using the PuTTY program.

The code for receiving the characters is almost same as the transmission program,Here we are enabling the receiver instead of the transmitter.

```
/*Enable receiver  */
UCSR0B = (1<<RXEN0);
```

The Atmega328p checks the status of the **RXC0 bit in UCSR0A** register continuously and waits for any data to be received.

24.12.2. USART Control and Status Register 0 A

Name: UCSR0A
Offset: 0xC0
Reset: 0x20
Property: -

Bit	7	6	5	4	3	2	1	0
	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0
Access	R	R/W	R	R	R	R	R/W	R/W
Reset	0	0	1	0	0	0	0	0

The **RXC0 bit is set** when a byte is received by the USART buffer.

```
while(1)
{
    while ( !(UCSR0A & (1<<RXC0)) ); /* Wait for data to be received */

    switch(UDR0) // using the switch() statement to  control the LED and Buzzer
    {
        case 'A' : DDRC  |= (1<<PC4); // Blinks LED
                   PORTC |= (1<<PC4);
                   _delay_ms(500);
                   PORTC &= ~(1<<PC4);
                   break;
```

```
            case 'B' : DDRC  |= (1<<PC5); // Beeps Buzzer
                       PORTC |= (1<<PC5);
                       _delay_ms(500);
                       PORTC &= ~(1<<PC5);
                       break;
             default  :
                       break;
    }
}
```

After the data is received a switch statement is used to select the appropriate action.

Full code is available in our Github repo.

If you want to do more than display the binary data on the screen,like you want to plot them or dump them into a database ,you should know how to read and write into serial port of a Windows or Linux Machine.

You can check out the following link to learn about Serial/Rs485 programming on Windows/Linux Systems.

- Collection of Serial/RS485 programming Tutorials

Tags:
ATmega328P embedded c avr Tutorial Serial Programming USB to RS485 Converter