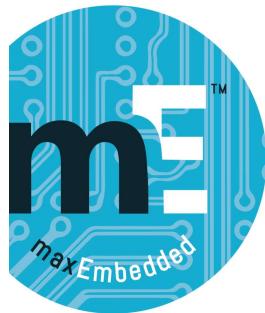


[maxEmbedded Index](#)[Categories »](#)[Tools](#)

Search This Site...

Shares



# Embedded

a guide      embedded      computer vision

[Home](#)[mE Index](#)[Getting Started](#)[Atmel AVR »](#)[Electronics](#)[More »](#)[About »](#)[Home](#)[Atmel AVR](#)[The USART of the AVR](#)

Search maxEmbedded

[Popular](#)[Recent](#)[Random](#)

MON

20

[The ADC of the AVR](#)

Posted by Mayank in Atmel AVR, Microcontrollers

FRI

24

[AVR Timers – TIMERO](#)

Posted by Mayank in Atmel AVR, Microcontrollers

TUE

06

[RF Module Interfacing without Microcontrollers](#)

Posted by Mayank in Electronics

THU

16

[LCD Interfacing with AVR](#)

Posted by Mayank in Atmel AVR, Microcontrollers

FRI

10

[I/O Port Operations in AVR](#)

Posted by Mayank in Atmel AVR, Microcontrollers



## The USART of the AVR

Hey folks! Lets continue with our tutorials on Serial Communication. Until now, we have seen a general [introduction to serial communication](#), followed by a tutorial on the concepts of [RS-232 communication](#). So here we are with the AVR communication protocols series, starting with the most basic ones, UART and USART! We will move on to SPI and TWI (I2C) later.



Some images used in this tutorial are taken directly from / and are a courtesy



Join our newsletter today for free.

Enter your email address

Subscribe Now



# Contents

- [UART and USART](#)

[USART Layout – How to set it up?](#)

Shares [USART Pin Configuration](#)

[Modes of Operation](#)

[Baud Rate Generation](#)

[Frame Formats](#)

- [Order of bits](#)
- [Number of Data bits](#)
- [Number of Stop bits](#)
- [Parity bits](#)
  - [Even/Odd Parity](#)
  - [Why use the Parity bit?](#)

[Register Description](#)

- [UDR, UCSRA, UCSRB, UCSRC, UBRR](#)

[Let's code it!](#)

- [Initializing UART](#)
- [Transmission/Reception Code](#)
- [Problem Statement – Example](#)
  - [Methodology](#)
  - [Hardware Connections](#)
  - [Coding](#)
  - [Video](#)
- [Can you do it?](#)
- [Summary](#)

Select Category 

Subscribe to maxEmbedded

Email Address

Subscribe

Like maxE on Facebook



MaxEmbe...  
4.1K followers

Follow Page

## UART and USART

The UART and USART have already been discussed [here](#). Anyways, lets have a quick recap.

UART stands for Universal Asynchronous Receiver/Transmitter. From the name itself, it is clear that it is asynchronous i.e. the data bits are not synchronized with the clock pulses.

USART stands for Universal Synchronous Asynchronous Receiver/Transmitter. This is of the synchronous type, i.e. the data bits are synchronized with the clock pulses.

If you refer to the USART section in the datasheet of any AVR microcontroller, you will find several features listed there. Some of the main features of the AVR USART are:

- Full Duplex Operation (Independent Serial Receive and Transmit Registers)
- Asynchronous or Synchronous Operation
- Master or Slave Clocked Synchronous Operation
- High Resolution Baud Rate Generator



Join our newsletter today for free.

Enter your email address

Subscribe Now



## USART Layout – How to set it up?

Before we continue, please note that the AVR USART is fully compatible with the AVR UART in terms of register bit locations, baud rate generation, transmitter/receiver operations and buffer functionality. So let us now have a quick look at how to set up USART in general. We will discuss in detail later.

1. The first step is to set the baud rate in both, the master and the slave. The baud rate has to be the same for both – master and slave.
2. Set the number of data bits, which needs to be sent.
3. Get the buffer ready! In case of transmission (from AVR to some other device), load it up with the data to be sent, whereas in case of reception, save the previous data so that the new received data can be overwritten onto it.
4. Then enable the transmitter/receiver according to the desired usage.

One thing to be noted is that in UART, there is no master or slave since master is defined by the MicroController, which is responsible for clock pulse generation. Hence Master and Slave terms occur only in the case of USART.

Master µC is the one which is responsible for Clock pulse generation on the Bus.

## USART Pin Configuration

Now lets have a look at the hardware pins related to USART. The USART of the AVR occupies three hardware pins pins:

1. RxD: USART Receiver Pin (ATMega8 Pin 2; ATMega16/32 Pin 14)
2. TxD: USART Transmit Pin (ATMega8 Pin 3; ATMega16/32 Pin 15)
3. XCK: USART Clock Pin (ATMega8 Pin 6; ATMega16/32 Pin 1)

## Modes of Operation

The USART of the AVR can be operated in three modes, namely-

1. Asynchronous Normal Mode
2. Asynchronous Double Speed Mode
3. Synchronous Mode

### Asynchronous Normal Mode

In this mode of communication, the data is transmitted/received asynchronously, i.e. we do not need (and use) the clock pulses, as well as the XCK pin. The data is transferred at the BAUD rate we set in the UBR register. This is similar to the UART operation.

### Asynchronous Double Speed Mode

This is higher speed mode for asynchronous communication. In this mode also we set the baud rates and other initializations similar to Normal Mode.



Shares

Join our newsletter today for free.

Enter your email address

Subscribe Now



The difference is that data is transferred at double the baud we set in the UBBR Register.

Setting the U2X bit in UCSRA register can double the transfer rate. Setting his bit has effect only for the asynchronous operation. Set this bit to zero when using synchronous operation. Setting this bit will reduce the divisor of the baud rate divider from 16 to 8, effectively doubling the transfer rate for asynchronous communication. Note however that the Receiver will in this case only use half the number of samples (reduced from 16 to 8) for data sampling and clock recovery, and therefore a more accurate baud rate setting and system clock are required when this mode is used. For the transmitter, there are no downsides.

## Synchronous Mode

This is the USART operation of AVR. When Synchronous Mode is used (UMSEL = 1 in UCSRC register), the XCK pin will be used as either clock input (Slave) or clock output (Master).

## Baud Rate Generation

The baud rate of UART/USART is set using the 16-bit wide UBRR register. The register is as follows:

Bit	15	14	13	12	11	10	9	8	
	URSEL	-	-	-	UBRR[11:8]				UBRRH UBRRL
	UBRR[7:0]								
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

UBRR Register (Click to Enlarge)

Since AVR is an 8-bit microcontroller, every register should have a size of 8 bits. Hence, in this case, the 16-bit UBRR register is comprised of two 8-bit registers – UBRRH (high) and UBRRL (low). This is similar to the 16-bit [ADC register](#) (ADCH and ADCL, remember?). Since there can be only specific baud rate values, there can be specific values for UBRR, which when converted to binary will not exceed 12 bits. Hence there are only 12 bits reserved for UBRR[11:0]. We will learn how to calculate the value of UBRR in a short while in this post.

The **USART Baud Rate Register (UBRR)** and the down-counter connected to it functions as a programmable prescaler or baud rate generator. The down-counter, running at system clock ( $F_{OSC}$ ), is loaded with the UBRR value each time the counter has counted down to zero or when the UBRRL Register is written. A clock is generated each time the counter reaches zero.

This clock is the baud rate generator clock output ( $= F_{OSC} / (UBRR + 1)$ ). The transmitter divides the baud rate generator clock output by 2, 8, or 16 depending on mode. The baud rate generator output is used directly by the receiver's clock and data recovery units.

Below are the equations for calculating baud rate and UBRR value:



Join our newsletter today for free.

Enter your email address

Subscribe Now

Shares

Operating Mode	Equation for Calculating BAUD Rate <sup>[1]</sup>	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

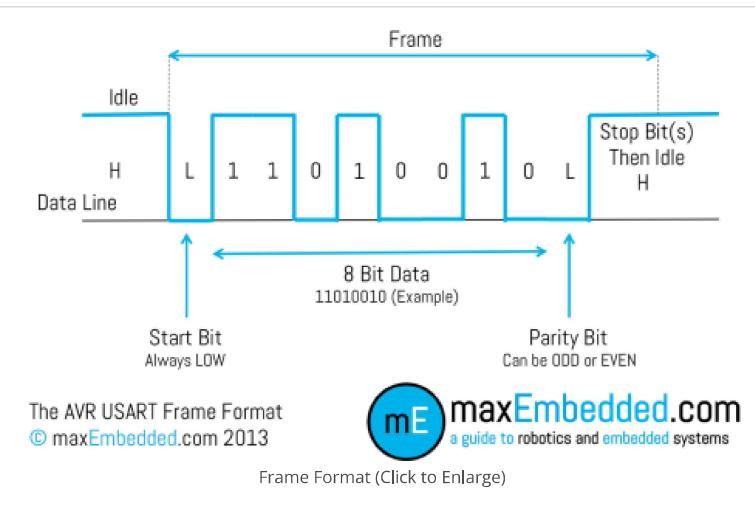
Baud Rate Calculation (Click to Enlarge)

1. BAUD = Baud Rate in Bits/Second (bps) (Always remember, Bps = Bytes/Second, whereas bps = Bits/Second)
2. FOSC = System Clock Frequency (1MHz) (or as per use in case of external oscillator)
3. UBRR = Contents of UBRRRL and UBRRRH registers

## Frame Formats

A frame refers to the entire data packet which is being sent/received during communication. Depending upon the communication protocol, the formats of the frame might vary. For example, TCP/IP has a particular frame format, whereas UDP has another frame format. Similarly in our case, RS232 has a typical frame format as well. If you have gone through the [loopback test](#) discussed in the [previous tutorial](#), you will notice that we have chosen options such as 8 bits data, 1 stop bit, no parity, etc. This is nothing but the selection of a frame format!

A typical frame for USART/RS232 is usually 10 bits long: 1 start bit, 8 data bits, and a stop bit. However a vast number of configurations are available... 30 to be precise!



## Order of Bits

1. Start bit (Always low)
2. Data bits (LSB to MSB) (5-9 bits)
3. Parity bit (optional) (Can be odd or even)
4. Stop bit (1 or 2) (Always high)

A frame starts with the start bit followed by the least significant data bit



Join our newsletter today for free.

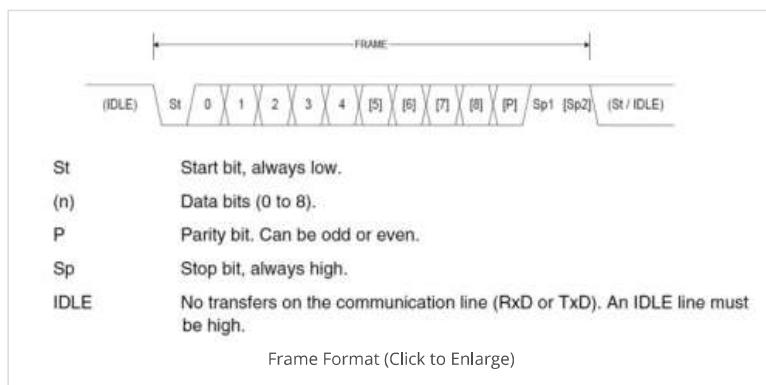
Enter your email address

Subscribe Now



the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, a new frame can directly follow it, or the communication line can be set to an idle ('high) state. Here is the frame format as mentioned in the AVR datasheet-

Shares



**Note:** The previous image (not the above one, the one before that) of Frame Format has a flaw in it! If you can find it, feel free to comment below! Let me see how many of you can spot it! And I'm not kidding, there *is* a mistake! ;)

## Setting the Number of DATA Bits

The data size used by the USART is set by the UCSZ2:0 bits in UCSRC Register. The Receiver and Transmitter use the same setting.

**Note:** Changing the settings of any of these bits (on the fly) will corrupt all ongoing communication for both the Receiver and Transmitter. Make sure that you configure the same settings for both transmitter and receiver.

UCSZ2	UCSZ1	UCS2:0	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

Data Bit Settings (Click to Enlarge)

## Setting Number of STOP Bits

This bit selects the number of stop bits to be inserted by the transmitter. *The Receiver ignores this setting.* The USBS bit is available in the UCSRC Register.

USBS	Stop Bit(s)
0	1-bit
1	2-bit

Stop Bit Settings (Click to Enlarge)

## Parity Bits

**Parity bits** always seem to be a confusing part. Parity bits are the simplest methods of error detection. Parity is simply the number of '1' appearing in



Join our newsletter today for free.

Enter your email address

Subscribe Now



the binary form of a number. For example, '55' in decimal is 0b00110111, so the parity is 5, which is odd.

Shares

## Even and Odd Parity

In the above example, we saw that the number has an odd parity. In case of even parity, the parity bit is set to 1, if the number of ones in a given set of bits (not including the parity bit) is odd, making the number of ones in the entire set of bits (including the parity bit) even. If the number of ones in a given set of bits is already even, it is set to a 0. When using odd parity, the parity bit is set to 1 if the number of ones in a given set of bits (not including the parity bit) is even, making the number of ones in the entire set of bits including the parity bit) odd. When the number of set bits is odd, then the odd parity bit is set to 0.

Still confused? Simply remember – even parity results in even number of 1s, whereas odd parity results in odd number of 1s. Lets take another example. 0d167 = 0b10100111. This has five 1s in it. So in case of even parity, we add another 1 to it to make the count rise to six (which is even). In case of odd parity, we simply add a 0 which will stall the count to five (which is odd). This extra bit added is called the parity bit! Check out the following example as well (taken from Wikipedia):

7 bits of data (count of 1 bits)		8 bits including parity	
		even	odd
0000000	0	00000000	00000001
1010001	3	10100011	10100010
1101001	4	11010010	11010011
1111111	7	11111111	11111110

## But why use the Parity Bit?

Parity bit is used to detect errors. Lets say we are transmitting 0d167, i.e. 0b10100111. Assuming an even parity bit is added to it, the data being sent becomes 0b101001111 (pink bit is the parity bit). This set of data (9 bits) is being sent wirelessly. Lets assume in the course of transmission, the data gets corrupted, and one of the bits is changed. Ultimately, say, the receiver receives 0b10001111. The blue bit is the error bit and the pink bit is the parity bit. We know that the data is sent according to even parity. Counting the number of 1s in the received data, we get four (excluding even parity bit) and five (including even parity bit). Now doesn't it sound amusing? There should be even number of 1s including the parity bit, right? This makes the receiver realize that the data is corrupted and will eventually discard the data and wait/request for a new frame to be sent. This is explained in the following diagram as well-



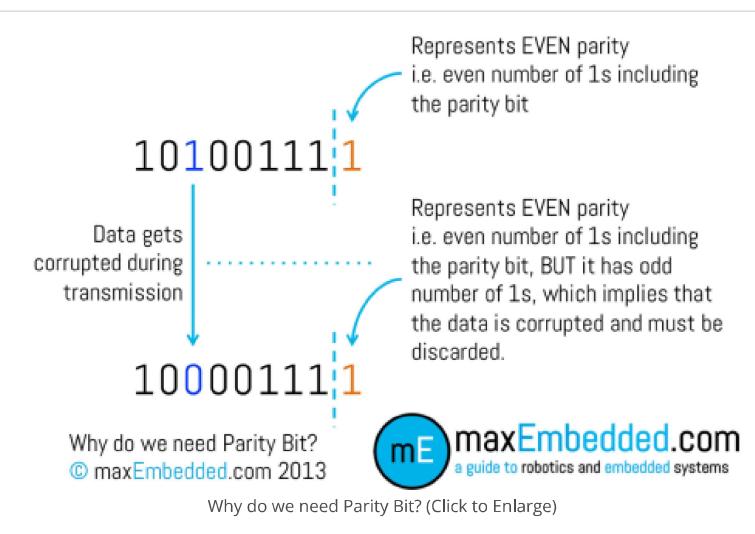
Join our newsletter today for free.

Enter your email address

Subscribe Now



Shares



Limitations of using single parity bit is that it can detect only single bit errors. If two bits are changed simultaneously, it fails. Using [Hamming Code](#) is a better solution, but it doesn't fit in for USART and is out of the scope of this tutorial as well!

The Parity Generator calculates the parity bit for the serial frame data. When parity bit is enabled ( $UPM1 = 1$ ), the Transmitter control logic inserts the parity bit between the last data bit and the first stop bit of the frame that is sent. The parity setting bits are available in the  $UPM1:0$  bits in the UCSRC Register.

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

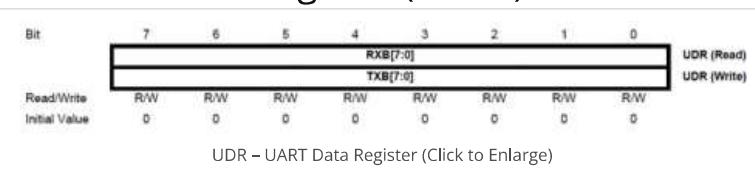
Parity Settings (Click to Enlarge)

Although most of the times, we do not require parity bits.

## Register Description

Now lets learn about the registers which deal with the USART. If you have worked with ADC and timers before, you would know that we need to program the registers in order to make the peripheral work. The same is the case with USART. The USART of AVR has five registers, namely UDR, UCSRA, UCSRB, UCSRC and UBBR. We have already discussed about UBBR earlier in this post, but we will have another look.

### UDR: USART Data Register (16-bit)



The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or



Join our newsletter today for free.

Enter your email address

Subscribe Now



written to the UDR Register location. Reading the UDR Register location will return the contents of the Receive Data Buffer Register (RXB).

For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

## JCSRA: USART Control and Status Register A (8-bit)

Bit	7	6	5	4	3	2	1	0	UCSRA
Read/Write	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	
Initial Value	R	R/W	R	R	R	R	R/W	R/W	0

UCSRA – USART Control and Status Register A (Click to Enlarge)

**Bit 7: RXC – USART Receive Complete Flag:** This flag bit is set by the CPU when there are unread data in the Receive buffer and is cleared by the CPU when the receive buffer is empty. This can also be used to generate a Receive Complete Interrupt (see description of the RXCIE bit in UCSRB register).

**Bit 6: TXC – USART Transmit Complete Flag:** This flag bit is set by the CPU when the entire frame in the Transmit Shift Register has been 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* (*yes, one and NOT zero*) to its bit location. The TXC Flag can generate a Transmit Complete Interrupt (see description of the TXCIE bit in UCSRB register).

- **Bit 5: UDRE – USART Data Register Empty:** The UDRE Flag indicates if the transmit buffer (UDR) is ready to receive new data. If UDRE is one, the buffer is empty, and therefore ready to be written. The UDRE Flag can generate a Data Register Empty Interrupt (see description of the UDRIE bit in UCSRB register). UDRE is set after a reset to indicate that the Transmitter is ready.
- **Bit 4: FE – Frame Error:** This bit is set if the next character in the receive buffer had a Frame Error when received (i.e. when the first stop bit of the next character in the receive buffer is zero). This bit is valid until the receive buffer (UDR) is read. The FE bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSRA.
- **Bit 3: DOR – Data Overrun Error:** 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 start bit is detected. This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.
- **Bit 2: PE – Parity Error:** This bit is set if the next character in the receive buffer had a Parity Error when received and the parity checking was enabled at that point (UPM1 = 1). This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.



Join our newsletter today for free.

Enter your email address

Subscribe Now



Shares

- Bit 1: U2X – Double Transmission Speed:** This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation. Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.

**Bit 0: MPCM – Multi-Processor Communication Mode:** This bit enables the Multi-processor Communication mode. When the MPCM bit is written to one, all the incoming frames received by the USART Receiver that do not contain address information will be ignored. The Transmitter is unaffected by the MPCM setting. This is essential when the receiver is exposed to more than one transmitter, and hence must use the address information to extract the correct information.

## JCSR: USART Control and Status Register B (8-bit)

Bit	7	6	5	4	3	2	1	0	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

UCSRB – USART Control and Status Register B (Click to Enlarge)

- Bit 7: RXCIE – RX Complete Interrupt Enable:** Writing this bit to one enables interrupt on the RXC Flag. A USART Receive Complete interrupt will be generated only if the RXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXC bit in UCSRA is set. The result is that whenever any data is received, an interrupt will be fired by the CPU.
- Bit 6: TXCIE – TX Complete Interrupt Enable:** Writing this bit to one enables interrupt on the TXC Flag. A USART Transmit Complete interrupt will be generated only if the TXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXC bit in UCSRA is set. The result is that whenever any data is sent, an interrupt will be fired by the CPU.
- Bit 5: UDRIE – USART Data Register Empty Interrupt Enable:** Writing this bit to one enables interrupt on the UDRE Flag (remember – bit 5 in UCSRA?). A Data Register Empty interrupt will be generated only if the UDRIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDRE bit in UCSRA is set. The result is that whenever the transmit buffer is empty, an interrupt will be fired by the CPU.
- Bit 4: RXEN – Receiver Enable:** Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxD pin when enabled.
- Bit 3: TXEN – Transmitter Enable:** Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled.



Join our newsletter today for free.

Enter your email address

Subscribe Now



- Bit 2: UCSZ2 – Character Size:** The UCSZ2 bits combined with the UCSZ1:0 bits in UCSRC register sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use. More information given along with UCSZ1:0 bits in UCSRC register.

Shares

**Bit 1: RXB8 – Receive Data Bit 8:** RXB8 is the ninth data bit of the received character when operating with serial frames with nine data bits. It must be read before reading the low bits from UDR.

**Bit 0: TXB8 – Transmit Data Bit 8:** TXB8 is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. It must be written before writing the low bits to UDR.

## JCSRC: USART Control and Status Register C (8-bit)

The UCSRC register can be used as either UCSRC, or as UBRRH register. This is done using the URSEL bit.

Bit	7	6	5	4	3	2	1	0	UCSRC
Read/Write	R/W								
Initial Value	1	0	0	0	0	1	1	0	
UCSRC – USART Control Register C (Click to Enlarge)									

- Bit 7: URSEL – USART Register Select:** This bit selects between accessing the UCSRC or the UBRRH Register. It is read as one when reading UCSRC. The URSEL must be one when writing the UCSRC.
- Bit 6: UMSEL – USART Mode Select:** This bit selects between Asynchronous and Synchronous mode of operation.

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

Synchronous/Asynchronous Selection (Click to Enlarge)

- Bit 5:4: UPM1:0 – Parity Mode:** This bit helps you enable/disable/choose the type of parity.

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

Parity Settings (Click to Enlarge)

- Bit 3: USBS – Stop Bit Select:** This bit helps you choose the number of stop bits for your frame.



Join our newsletter today for free.

Enter your email address

Subscribe Now



USBS	Stop Bit(s)
0	1-bit
1	2-bit

Stop Bit Settings (Click to Enlarge)

Shares

**Bit 2:1: UCSZ1:0 – Character Size:** These two bits in combination with the UCSZ2 bit in UCSRB register helps choosing the number of data bits in your frame.

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

Data Bit Settings (Click to Enlarge)

- Bit 0: UCPOL – Clock Polarity:** This bit is used for Synchronous mode only. Write this bit to zero when Asynchronous mode is used. The UCPOL bit sets the relationship between data output change and data input sample, and the synchronous clock (XCK).

UCPOL	Transmitted Data Changed (Output of TxD Pin)	Received Data Sampled (Input on RxD Pin)
0	Rising XCK Edge	Falling XCK Edge
1	Falling XCK Edge	Rising XCK Edge

UCPOL Bit Settings (Click to Enlarge)

## UBRR: USART Baud Rate Register (16-bit)

Bit	15	14	13	12	11	10	9	8	UBRRH UBRRL
	URSEL	-	-	-	UBRR[11:8]	UBRR[7:0]			
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	UBRRH UBRRL
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

UBRR Register (Click to Enlarge)

We have already seen this register, except the URSEL bit.

- Bit 15: URSEL:** This bit selects between accessing the UBRRH or the UCSRC Register. It is read as zero when reading UBRRH. The URSEL must be zero when writing the UBRRH.

## Let's code it!

Now its been enough of theory. Let's get our hands dirty with some actual code! Here is how we will structure out code – first initialize UART, then read



Join our newsletter today for free.

Enter your email address

Subscribe Now



## Initializing UART

Initializing UART involves the following steps:

- Shares
1. Setting the Baud Rate
  2. Setting Data Size (5/6/7/8/9 bits)
  3. Enabling Reception/ Transmission (In the TXEN and RXEN bits in UCSRB)
  4. Setting parity, and number of Stop Bits.

Below is the code to initialize UART, explanation follows it. You can scroll the code sideways in order to view it completely.

```

1 // define some macros
2 #define BAUD 9600
3 #define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)           // define
4
5 // function to initialize USART
6 void uart_init (void)
7 {
8     UBRRH = (BAUDRATE>>8);                      // shift the i
9     UBRRL = BAUDRATE;                             // set baud ra
10    UCSRB|= (1<<TXEN)|(1<<RXEN);             // enable rece
11    UCSRC|= (1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1); // 8bit data + 1
12 }
```

## Explanation

The variable (or macro) used to set the baud rate is `BAUD`. The variable (or macro) `BAUDRATE` is defined as per the calculations according to which, the baud rate is set in the registers.

One thing to note is that `UCSRC` and `UBRRH` share the same I/O location in the memory. So, the bit that controls the selection/switching of the registers is the `URSEL` Bit in the `UCSRC` register. This creates a lot of confusion, and if this bit is not handled properly, UART doesn't work! So pay attention to the code initialization.

If `URSEL` is zero during a write operation, the `UBRRH` value will be updated. If `URSEL` is one, the `UCSRC` setting will be updated. This means that while `URSEL` bit is 0, and even if we are addressing `UCSRC`, `UBBRH` will be addressed, which would in turn result in setting up of wrong, non standard baud rates, and everything getting messed up! Since the default value of `URSEL` is 0, it is safe to first use `UBRRH`, and then use the `UCSRC` Register.

The bits `TxEN` and `RxEN` are set to 1 to enable UART transmission and reception.

The Bits `UCSZ0` and `UCSZ1` are used to select the number of data bits to be transmitted. As per the table, we have set the two bits to transmit/receive 8 bits.

## Transmission/Reception Code

The following code is for transmitting data, explanation follows it. You can scroll the code sideways in order to view the code completely.



Join our newsletter today for free.

Enter your email address

Subscribe Now



```

1 // function to send data
2 void uart_transmit (unsigned char data)
3 {
4     while (!(UCSRA & (1<<UDRE)));           // wait while
5     UDR = data;                                // load data
Shares 6 }
```

The following code is for transmitting data, explanation follows it. You can scroll the code sideways in order to view to code completely.

```

1 // function to receive data
2 unsigned char uart_recieve (void)
3 {
4     while(!(UCSRA) & (1<<RXC));           // wait while
5     return UDR;                            // return 8-bit
6 }
```

## Explanation

The above code snippets are to transmit and receive data respectively. In both of them, we first wait for the flag bits denoting data register empty bit UDRE, and receive complete RXC bit respectively to get set, and then take the next action.

When we transmit the data, we load the 8-bit data into the UDR (UART Data Register) to be transmitted.

When we want to receive data, we wait for the whole data to be received first, and then return the value of the received data stored in UDR.

## Problem Statement

Let's take a problem statement now. Let's say we have to send serial data through a serial terminal emulator on a PC, and receive it on a AVR microcontroller, and display it on an LCD. The displayed values should be in ASCII format.

## Methodology

For this, we need to connect an LCD to our microcontroller, and configure the LCD header file accordingly. To learn about interfacing LCD with AVR, read [this tutorial](#) written by Mayank. We will also need to connect the [USB-UART Bridge](#) with the microcontroller. If you have followed the [loopback test](#) of the previous tutorial, then this will be easy for you. Here is the methodology you need to follow:

1. Send data from your PC using a serial terminal emulator. This has been discussed in the [loopback test](#).
2. Connect your microcontroller to your PC using the USB-UART Bridge. Schematics follow.
3. Connect your LCD with your microcontroller. Schematics follow.
4. Receive serial data through the RxD pin of your microcontroller.
5. Process the data.



Join our newsletter today for free.

Enter your email address

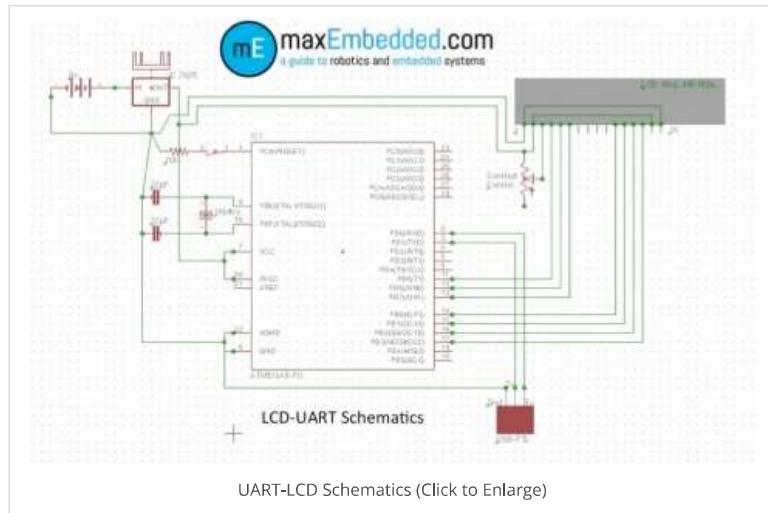
Subscribe Now



Shares

## Hardware Connections

Here is the schematics which shows how to connect LCD and USB-UART bridge with your microcontroller. Remember, the TX of USB-UART bridge is connected to the RX of the microcontroller, and vice versa.



## Coding

I have made some changes in the LCD header file (lcd.h), so that the LCD is operated by 2 ports (PORTB and PORTD) instead of a single port. Following are the changes: (refer to [this](#) tutorial to know more)

```

1 #define LCD_PORT1      PORTD      /**< port1 for the LCD
2 #define LCD_PORT2      PORTB      /**< port2 for the LCD
3 #define LCD_DATA0_PORT LCD_PORT2  /**< port for 4bit data
4 #define LCD_DATA1_PORT LCD_PORT2  /**< port for 4bit data
5 #define LCD_DATA2_PORT LCD_PORT2  /**< port for 4bit data
6 #define LCD_DATA3_PORT LCD_PORT2  /**< port for 4bit data
7 #define LCD_DATA0_PIN   0          /**< pin for 4bit data
8 #define LCD_DATA1_PIN   1          /**< pin for 4bit data
9 #define LCD_DATA2_PIN   2          /**< pin for 4bit data
10 #define LCD_DATA3_PIN  3          /**< pin for 4bit data
11 #define LCD_RS_PORT    LCD_PORT1 /**< port for RS line
12 #define LCD_RS_PIN     5          /**< pin for RS line
13 #define LCD_RW_PORT    LCD_PORT1 /**< port for RW line
14 #define LCD_RW_PIN     6          /**< pin for RW line
15 #define LCD_E_PORT     LCD_PORT1 /**< port for Enable line
16 #define LCD_E_PIN      7          /**< pin for Enable line

```

What I am going to demonstrate here is very simple. We just need to configure LCD, UART, and its done! Below is the full code. The code is commented well, and doesn't need any further explanation (provided you have understood all the concepts till now, which I truly believe!). In case you need any help in comprehending the code, feel free to comment below. You can scroll the code sideways in order to view to code completely.

## Full Code

You can also find the code on [pastebin](#), and the [AVR code gallery](#). The AVR code gallery contains the LCD files as well. To learn how to write, build,



Join our newsletter today for free.

Enter your email address

Subscribe Now



[+ expand source](#)

## Hardware Setup

Shares My hardware setup looked something like this-



AVR USART Hardware Setup (Click to Enlarge)

## Video

Here is a short video of the implementation of the above code.

AVR USART - Sending data from PC to microcontroller



## Can you do it?

Alright, so this was just an introduction. You can try out the following things as well by modifying the above code:

1. Extend the above problem statement by sending back the data you received from the computer with some alterations. For example, if you send maxEmbedded from your PC, you should receive [m] [a] [x] [E] [m] [b] [e] [d] [d] [e] [d] back from the microcontroller, which should be displayed in the serial terminal console window.

**Hint:** While sending the received data, send it in between [ and ].



Join our newsletter today for free.

Enter your email address

Subscribe Now



Shares

- a. Receive Data
- b. Send '['
- c. Send received Data back
- d. Send ']'
- e. Repeat this forever

2. Send data from one microcontroller to another microcontroller, and display the result using an LCD.

**Hint:** You should connect the TXD of one microcontroller to RXD of another microcontroller. You can use wired connection for now.

3. Extend the above to a full-duplex communication. Two microcontrollers send and receive data simultaneously.

**Hint:** Connect TXD of one microcontroller to RXD of other. You can use wired connection for now.

## Do it and show the world that you are awesome!

If you have tried any of the above three challenges, then you are most welcome to record a video of your implementation, post it on YouTube and share the link below as comment! This will ensure that the world looks at your awesomeness. But wait, its not over yet! Your video might get featured just below this line as well! So, what are you waiting for?

## Summary

Now that we have come to an end of another long post, let's summarize what we have learnt-

- We know what's UART and USART.
- We know how to set up USART, its pin configuration and three modes of operation.
- We learnt how to set baud rate for a USART data transfer.
- We learnt how to send the data – frame formats, data bits, stop bits, and about parity.
- We got familiarized with different USART registers – UDR, UCSRA, UCSRB, UCSRC and UBRR.
- We saw some sample codes, and then implemented a problem statement as well.
- We learnt how to make hardware connections as well.

**So folks, that's it for now. We will return back soon with more tutorials!**

**:) So subscribe to stay updated! And don't forget to share your views and questions as comments below! :)**

### Written By-

*Yash Tambi*

*VIT University, Vellore*

*tambi@maxEmbedded.com*

*Mayank Prasad*

*Arizona State University*

*max@maxEmbedded.com*

Join our newsletter today for free.

Enter your email address

Subscribe Now



## 126 Comments

Shares

Enter Comments

Chazz October 14, 2016

Hello max.. can u explain why USART starts communication (tx), with lsb as first why not msb first.. not asking about avr generally it states in USART like b first..

thanks alot..

[REPLY](#)

hanieh November 15, 2016

Hi dear sir.

I want to use uart comunication between 2 avrs.like at mega 32.and i want send and receive by interrupt.can you help me please?

[REPLY](#)



aditpendam December 22, 2016

Would this code work on an attiny841, without the lcd library?

[REPLY](#)



Pratik Khetan March 7, 2017

Hi,

I am using ATmega328p for a project where I would be needing two serial ports. How can I configure it using Embedded C to have a Software Serial as well.

[REPLY](#)



Join our newsletter today for free.

Enter your email address

Subscribe Now



It would be much nicer if you add these macros in your example:

```
#define SET_BIT(input, n) ((input) |= (1 << (n)))
Shares define CLEAR_BIT(input, n) ((input) &= ~(1 << (n)))
define TOGGLE_BIT(input, n) ((input) ^= (1 << (n)))
define CHECK_BIT(input, n) ((input) & (1 << (n)))
```

[REPLY](#)
 john June 25, 2017

ey, mayank

i'm trying to receive the data from bluetooth and displaying it on lcd. But when i press 1 from the android app (bluetooth Ssp pro), i receive something else but not 1. I used the same technique used here. UDR is copied to some variable and converted using itoa but still no luck. Please comment. I am doing this using atmega 8.

[REPLY](#)


geeth August 10, 2017

hi,I tried the following code in AVR studion and when i check the output through cool term I get a hex value'80' instead of 75 which is the hex value of 'k'

```
// define some macros
#define F_CPU 8000000UL
#include
#include
#define BAUD 9600 // define baud
#define BAUDRATE 51 // set baud rate value for UBRR

// function to initialize UART
void uart_init (void)
{
    UBRR0H = (BAUDRATE>>8); // shift the register right by 8 bits
    UBRR0L = BAUDRATE; // set baud rate
    UCSR0B |= (1<<TXEN0)|(1<<RXEN0); // enable receiver and transmitter
    UCSR0C |= (1<<UCSZ00)|(1<<UCSZ01); // 8bit data format
}
// function to send data
void uart_transmit (unsigned char data)
{
```



Join our newsletter today for free.

[Subscribe Now](#)



```

UDR0 = 4; // load data in the register
_delay_ms(1000);
}
// function to receive data
Shares signed char uart_receive (void)

while(!(UCSR0A) & (1<<RXC0)); // wait while data is being received
return UDR0; // return 8-bit data

t main()

uart_init();
char data='k';
while(1)

    uart_transmit(data);
    uart_receive();

}

```

[REPLY](#)

Rishabh Gupta May 2, 2018

Hello Mayank,  
 I am trying to transfer a character from USART . As I am not having an LCD right now so I am trying to blink a LED when 1 is passed. But it is not working . I am using UNO CH 340 (Atmega 328p) . can you help me???

[REPLY](#)

Krazitcek August 29, 2018

Hello, you said the default value of URSEL is 0, how do you know that. I try to understand with atmega8 datasheet and i see in usart registers section that for UCSRC register initial value is 1 and for UBRRH initial value is 0, so as it is the same bit, i don't understand what is the default value !

[REPLY](#)

ahmad stout September 19, 2018



Join our newsletter today for free.

Enter your email address

Subscribe Now



Hello, Thank you for your tutorial  
im confusing with the code i try more than way to transmit and recive data  
from to avr i use atmega16 and i use the same code you but it with usb to

Shares

nally all what i got from avr a (Question Mark ???? Or unkown words)  
in you please help me?

[REPLY](#)

Michael February 1, 2019

ell, the above proposed program has not been updated for a long time,  
nd it makes no sense to use it for current purposes. The use of two  
ograms that would apply simultaneously to the same port is impossible,  
is impossible to do without a third-party program. It is now easier with  
this, many programs that create virtual serial port driver and link them. On  
the example of a high-voltage network, there is an article on how to use it  
in a similar situation <https://www.virtual-serial-port.org/article/avr-and-pc-communication/>

[REPLY](#)



Robert Yahir May 14, 2019

Please can you also provide code in assembly language.

[REPLY](#)



mahbub June 25, 2019

when i use UCSR0C or UCSR0A or same type of other pin show an error  
{Symbol 'UCSR0A' could not be resolved}. How to solve this error

[REPLY](#)



Atahan September 3, 2019



Join our newsletter today for free.

Enter your email address

Subscribe Now



Hi there

In the following function one thing makes me confused, "UBRRH=

AUDRATE>>8);"

Shares

hat was it intended to use with this code

/ the way your tutorial is so good and so useful.

Thank you from now

[REPLY](#)

Dervabaw Dese June 21, 2020

Regarding to Data frame, I tried to figure out where the mistake was – It is possible that the error is in the order of the transfer of data, i.e the LSB send goes out first and then the rest of the bits up to the MSB?

[REPLY](#)

[← Older Comments](#)

## Trackbacks/Pingbacks

[UART y USB en Arduino | Aprendiendo Arduino](#) - [...]

<http://maxembedded.com/2013/09/the-usart-of-the-avr/> [...]

[Software serial: como fazer? - Embarcados](#) - [...] Figura 1: Frame de comunicação serial. Imagem capturada da Maxim Integrated através do link: <http://maxembedded.com/2013/09/the-usart-of-the-avr/> [...]

[AVR USART Tutorial \(PART I\) | Embedded System Tutorials](#) - [...] has 1 USART port. For detail information about ATmega8 USART have a look at Datasheet or this tutorial. ATmega8...

[IoT Demo 2k18: The Head Unit Firmware – Da Vinci Space](#) - [...] slave. Whereas on hardware design level, there was connected an RS485 shield as necessary to the USART (=the serial...)

[Микроконтроллеры AVR: пример работы с часами реального времени DS1302 | Записки программиста](#) - [...] (исходники к книге лежат здесь), а также из статьи The USART of the AVR в блоге maxembedded.com. А вот...

## Leave a Reply



Join our newsletter today for free.

Enter your email address

Subscribe Now



Enter your comment here...

Shares

The site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

## Copyright



maxEmbedded by [Mayank Prasad](#)  
is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License](#).

© Copyright 2011-2014 by  
maxEmbedded. Some rights  
reserved.

## Tags

accelerometer adapter adc assembly avr  
**adc** **avr** **basics** avr eeprom avr  
 interrupts avr serial avr timers  
**basics** code gallery code optimization  
 color spaces **counters** cv basics  
 development server filter circuit i2c iot basics ir  
 sensor matlab msp430 basics msp430  
 launchpad opencv pcb design pic basics  
 power supply printed circuit boards pwm  
**raspberry pi** rf communication rf  
 module robotics robot locomotion rs232  
**sbc** **basics** **sensors** **Serial** **spi**  
**timers** uart usart web development  
 wireless

## Top Posts & Pages

[AVR Timers - TIMER0](#)

[AVR Timers - CTC Mode](#)

[The ADC of the AVR](#)

[The USART of the AVR](#)

[AVR Timers - TIMER1](#)

[Introduction to AVR Timers](#)

[Setting up AVR-GCC Toolchain on Linux and Mac OS X](#)

[Home](#)

[Copyright](#)

[Privacy Policy](#)

[Credits](#)

[Sitemap](#)

[Contact](#)

© Copyright 2011-2014 by maxEmbedded. Some rights reserved.

maxEmbedded is a free and open source platform to share knowledge and learn about the concepts of robotics, embedded systems and computer vision. People from



Join our newsletter today for free.

Enter your email address

Subscribe Now



around the world who are enthusiastic about these topics and willing to support the open source community are invited to share their information, knowledge and expertise by means of written tutorials and videos on the website.

Shares