Microcontrollers

USART – Serial communication

Christiaen Slot – c.g.m.slot@saxion.nl
Hans Stokkink – j.s.d.stokkink@saxion.nl

SAXION

Kom verder. Saxion.

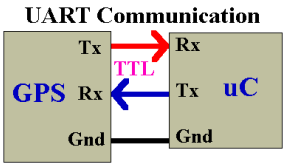
Content

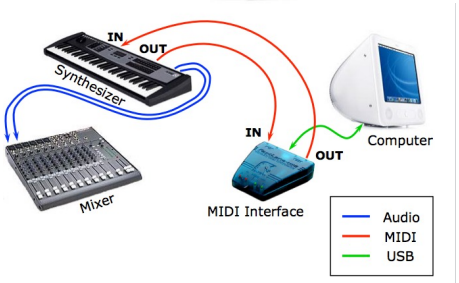
- What is serial communication?
- Asynchronous and Synchronous communication
- Atmega USART overview
- Example how to use USART

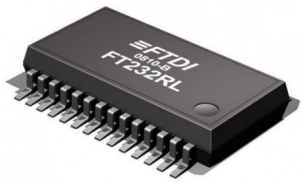
Definition & Examples

- In telecommunication and computer science, **serial communication** is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus.

- "U(S)ART"
- Morse code telegraphy
- Ethernet
- I2C , SPI
- DMX512 (theatrical lighting)
- MIDI (control of electronic musical instruments)
- RS-232, RS-485
- Universal Serial Bus (USB)
- PCI Express
- FireWire
- Serial Attached SCSI

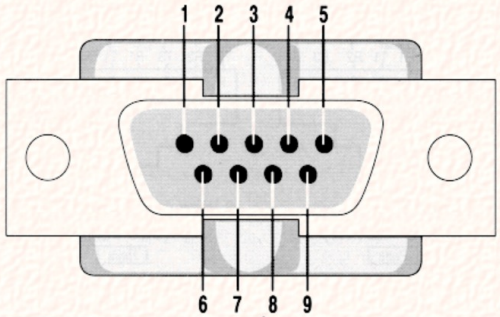






Serial communication hardware RS232

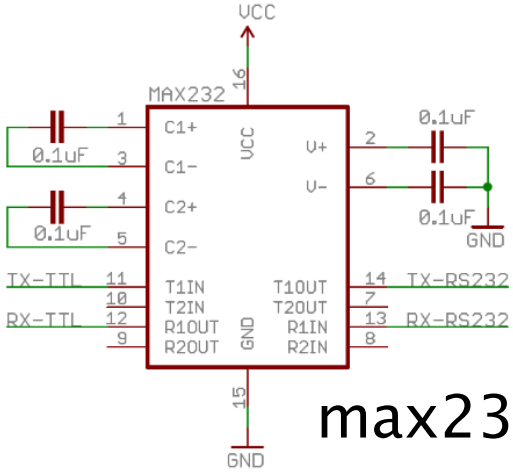
used for Asynchronous Data



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

RS232 logic and voltage levels

Data circuits	Control circuits	Voltage
0 (space)	Asserted	+3 to +15 V
1 (mark)	Deasserted	-15 to -3 V



max232

Communication speed

Baud rate = symbols per second

Serial ports use two-level (binary) signaling, so the data rate in bits per second is equal to the symbol rate in baud

Baud/bit rates commonly supported include 75, 110, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600 and 115200 bit/s

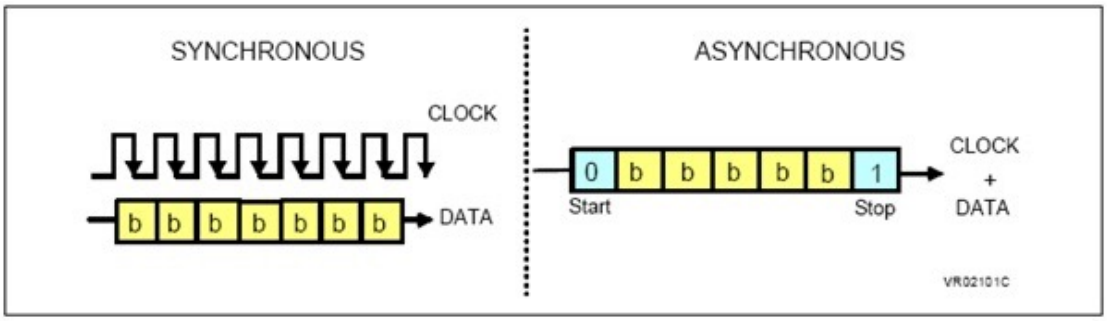
ASCII

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

Content

- What is serial communication?
- **Asynchronous and Synchronous communication**
- Atmega USART overview
- Example how to use USART

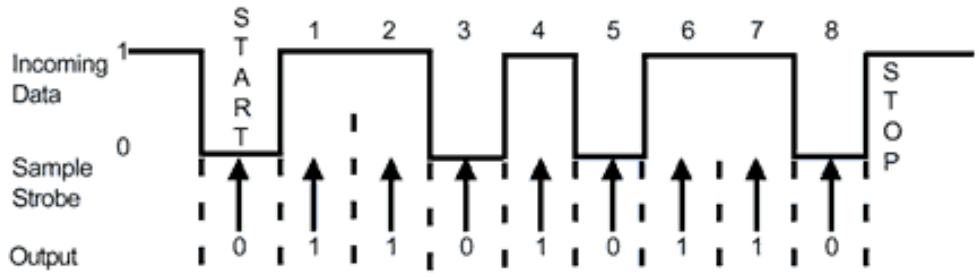
Asynchronous and Synchronous serial communication



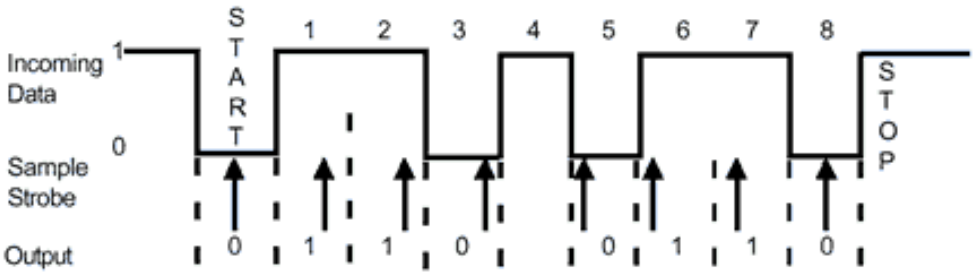
Asynchronous and Synchronous serial communication advantages an disadvantages

	Advantages	Disadvantages
Asynchronous transmission	• Simple, doesn't require synchronization of both communication sides • Cheap, because asynchronous transmission requires less hardware • Setup is faster than other transmissions, so well suited for applications where messages are generated at irregular intervals, for example data entry from the keyboard, and the speed depends on different applications.	• Large relative overhead, a high proportion of the transmitted bits are uniquely for control purposes and thus carry no useful information
Synchronous transmission	• Lower overhead and thus, greater throughput	• Slightly more complex • Hardware is more expensive

Asynchrous data sampling



a. Best case, receiver samples at midpoint of each bit.



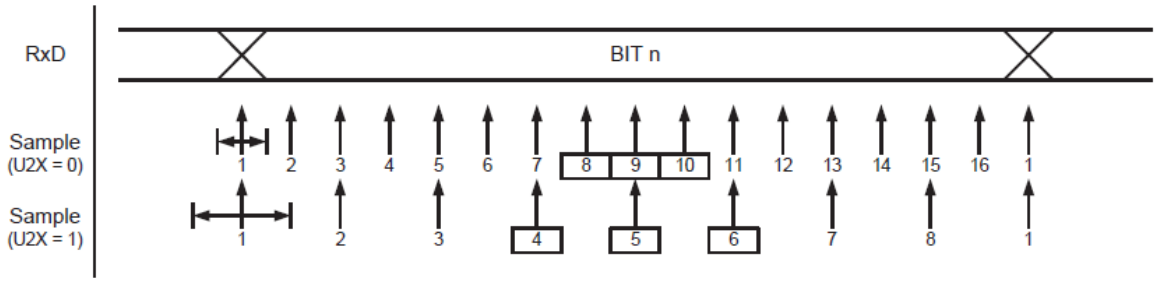
b. Receiving clock is too slow, causing bit 4 to be skipped and the data to be corrupted.

Ideal and corrupted asynchronous data sampling

's

Asynchronous Data Recovery

Figure 19-6. Sampling of Data and Parity Bit



The decision of the logic level of the received bit is taken by doing a majority voting of the logic value to the three samples in the center of the received bit.

Sampling rate = 8x or 16x baudrate

Content

- What is serial communication?
- Asynchronous and Synchronous communication
- **Atmega USART overview**
- Example how to use USART

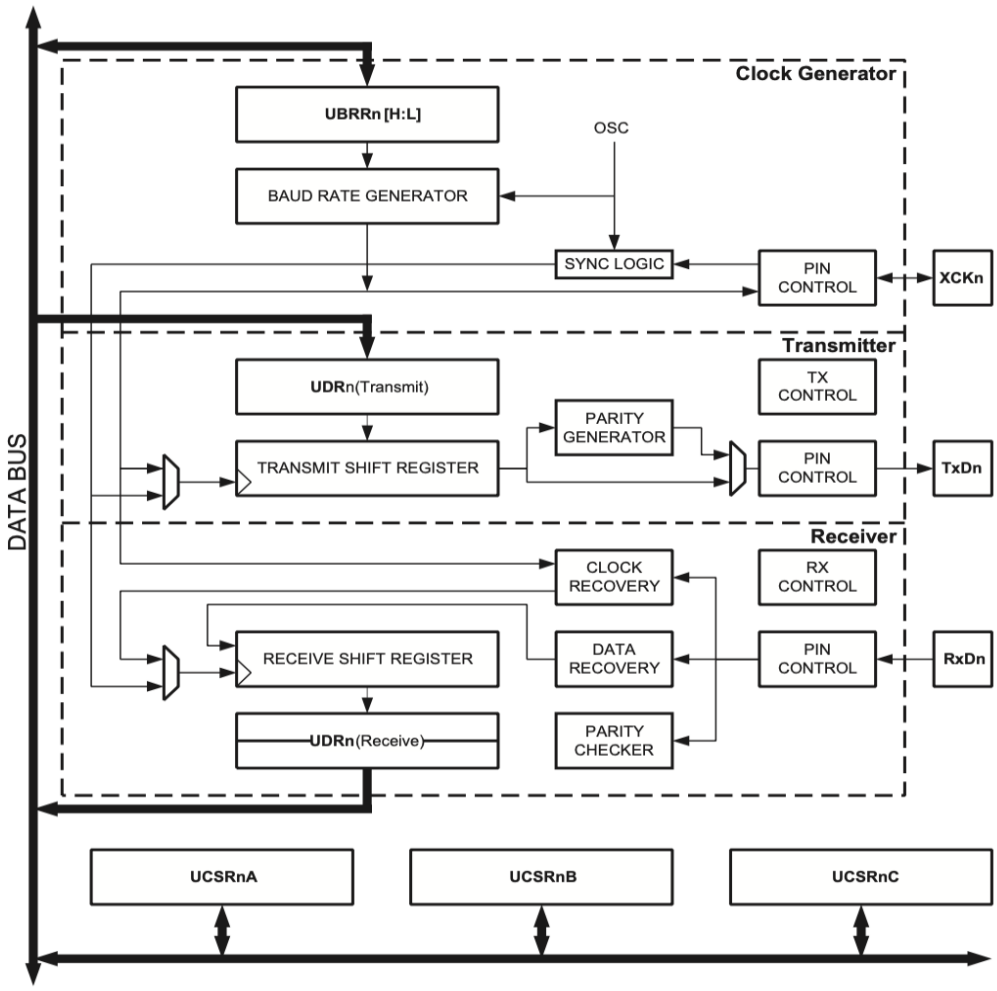
ATmega features

Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART)

As seen in the datasheet:

- **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**

USART block diagram



USART clock generator & Baudrate

Figure 19-2. Clock Generation Logic, Block Diagram

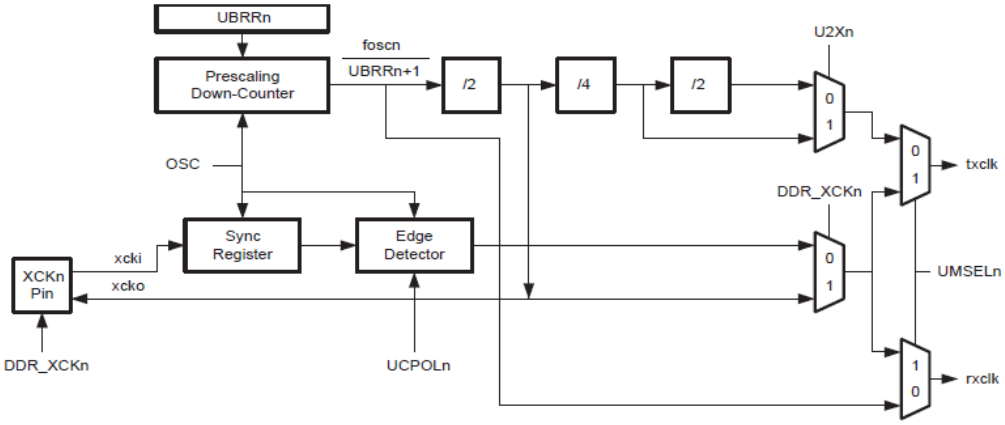
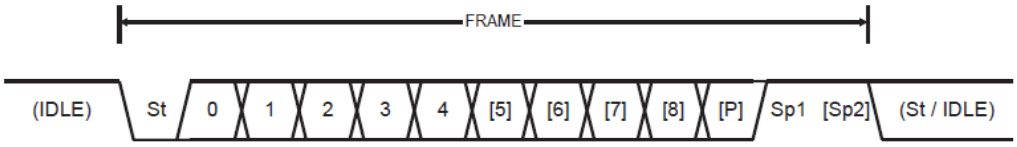


Table 19-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRRn Value
Asynchronous Normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(\boldsymbol{UBRR}n + 1)}$	$\boldsymbol{UBRR}n = \frac{f_{OSC}}{16BAUD} - 1$

USART Frame Formats

Figure 19-4. Frame Formats



St Start bit, always low.

(n) Data bits (0 to 8).

P Parity bit. Can be odd or even.

Sp Stop bit, always high.

IDLE No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.

The frame format used by the USART is set by the UCSZn2:0, UPMn1:0 and USBSn bits in UCSRnB and UCSRnC.

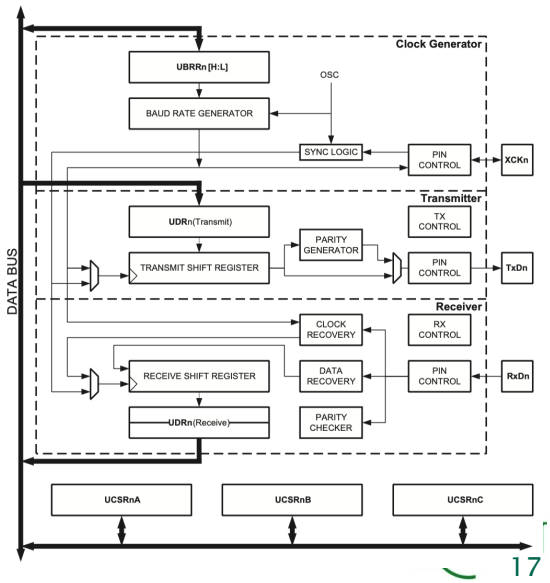
USART flags

Transmitter flags:

- USART Data Register Empty (UDREn)
- Transmit Complete (TXCn)

Receiver flags:

- Receive Complete (RXCn)
- Frame Error (FEn)
- Data OverRun (DORn)
- Parity Error (UPEn)



Content

- What is serial communication?
- Asynchronous and Synchronous communication
- Atmega USART overview
- **Example how to use USART**

skeleton code example

```
// calculate and fill in the correct baudrate for 9600 on 16 MHz
#define baudrate ...

void initUART() {
    // set UART baudrate
    UBRR0 = baudrate;

    // Look up & set correct frame format: 8data, 2stop bit, no parity
    UCSR0C = (...<<UPM01) | (...<<UPM01)| (...<<USBS0) | (...<<UCSZ02)| (...<<UCSZ01)|

    // enable rx & tx & interrupt receive<CR>
    UCSR0B = (1<<RXEN0) | (1<<TXEN0) | (1<<RXCIE0) ; // uart runs from now on.
};

void UARTsendchar(unsigned char data) {
    while ( ... ) {
    // wait till UDR buffer is empty before filling
    }
    UDR0 = data;
}

unsigned char USARTreceivechar( void ) {

    while ( !(UCSR0A & ( 1 << RXC0)) ) {
        // wait for data to be received;
    }
    return UDR0;
    // get and return received data from buffer
}
```

poll yourself

skeleton code example

```
// calculate and fill in the correct baudrate for 9600 on 16 MHz
#define baudrate ...

void initUART() {
    // set UART baudrate
    UBRR0 = baudrate;

    // Look up & set correct frame format: 8data, 2stop bit, no parity
    UCSR0C = (...<<UPM01) | (...<<UPM01)| (...<<USBS0) | (...<<UCSZ02)| (...<<UCSZ01)|

    // enable rx & tx & interrupt receive<CR>
    UCSR0B = (1<<RXEN0) | (1<<TXEN0) | (1<<RXCIE0) ; // uart runs from now on.
};

void UARTsendchar(unsigned char data) {
    while ( ... ) {
    // wait till UDR buffer is empty before filling
    }
    UDR0 = data;
}

ISR(USART_RX_vect) {
    // when receive something, put it on the leds
    // except when cr or lf
    char temp = UDR0;
    if (!((temp==0x0D) || (temp==0x0A))) { // filter out lf & cr
        PORTB = temp;
        latestChar = temp; // store received byte
    }
}
```

interrupt