INTERFACING

UART

AMIT

Communication Protocol:

Concept of Communication Protocol:

- Communication Protocol:
  it is a protocol to communicate between two ECUs or more that can make a network between of them.

- Network:
  it exists when there are two or more ECU communicate between of themselves.

- Electrical Controller Unit:
  it is microcontroller or more with some of additional component that make a single node into the network.

Communication Protocol:

Specs of Communication Protocol:

- **Medium**:

  it can be wired or wireless,

Criteria	Wired	Wireless
Speed	Higher	Lower
Connection	Complex	Simple
Safety	More safe	It is safe, but none ensures
Security	It is more security	It is easy to hack
Implantation	It is very easy	It is very difficult
Mobility	It is very difficult	It is easy
Cost	It has a low cost	It highly costs

because the security of wired is higher than wireless, wired is commonly used.

Communication Protocol:

Specs of Communication Protocol:

- Transmission Techniques:
  - Parallel:
    - it is high speed, but it is very complex.
    - it may cause a back EMF because of huge number of wires.
    -it causes a data skew, so a delay must be used to ensure that all of bits arrive, so the speed will become lower.
  - Series:
    - it is very simple to implement.
    -it is slower than parallel, but it has an enough speed for transferring.
  - Series communication is commonly and widely used in embedded systems.

Communication Protocol:

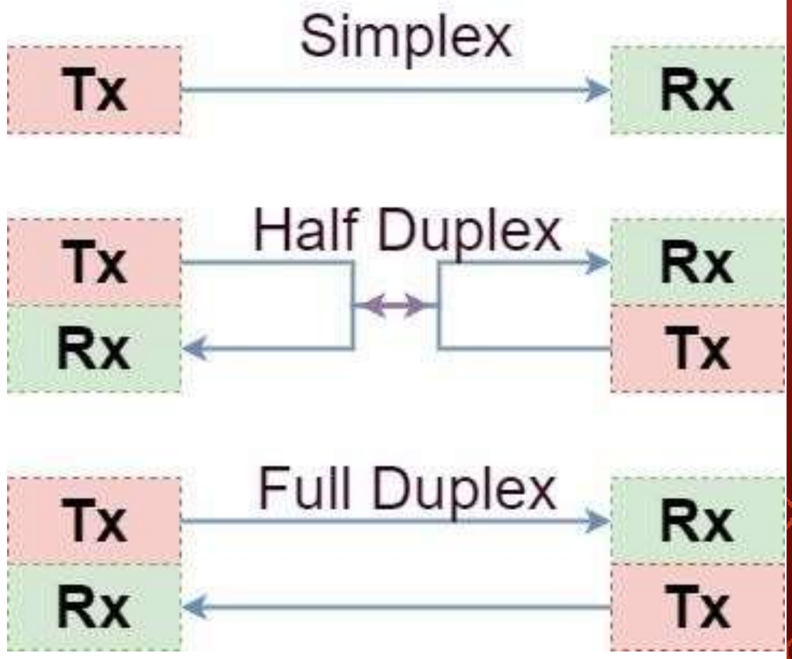
Specs of Communication Protocol:

- Data Direction:
  - Simplex:

    - data has only one direction from transmitter to receiver.
  - Half Duplex:

    -data has two directions from transmitter to receiver and vice versa, but only one direction runs per time.
  - Full Duplex:

    -data has two directions from transmitter to receiver and vice versa, but here every direction can run separately.



Communication Protocol:

Specs of Communication Protocol:

- Synchronization:
    - Asynchronized:
      -that means every node into the network has a separated clock for it, so the communication protocol must depend on the baud rate, means depending on same speed rate of transferring data.
      - there is no clock wire into the network by 100%.
    - Synchronized:
      -that mean the transferring of data depends on shared clock that comes out of the transmitter node and feeds the receiver node.
      - - there is a clock wire into the network by 90%, why not 100%?.

Communication Protocol:

Specs of Communication Protocol:

- Receiver/Transmitter Relationship:
  - Peer to Peer:

    - max nodes are two, Like ***UART***.
  - Single Master/Single Slave "**SMSS**".
  - Single Master/Multi Slaves "**SMMS**":

    - like ***SPI***
  - Multi Masters/No Slaves "**MMNS**":

    - like ***CAN***
  - Multi Masters/Single Slave "**MMSS**":
  - Multi Masters/Multi Slaves "**MMMS**":

    - Like ***IIC***.

Communication Protocol:

Specs of Communication Protocol:

- Data Throughput:
  it means the pure data or the useful data per all frame.
  it will be declared in details later.

$$T\square roug\square put\ (Tp) = \frac{Useful\ Bits\ of\ Data}{All\ Bits\ of\ Frame}$$

AMIT

Communication Protocol

Specs of Communication protocol

- Digital Level:

  - there are two techniques of transistors

    - TTL.

    - CMOS.

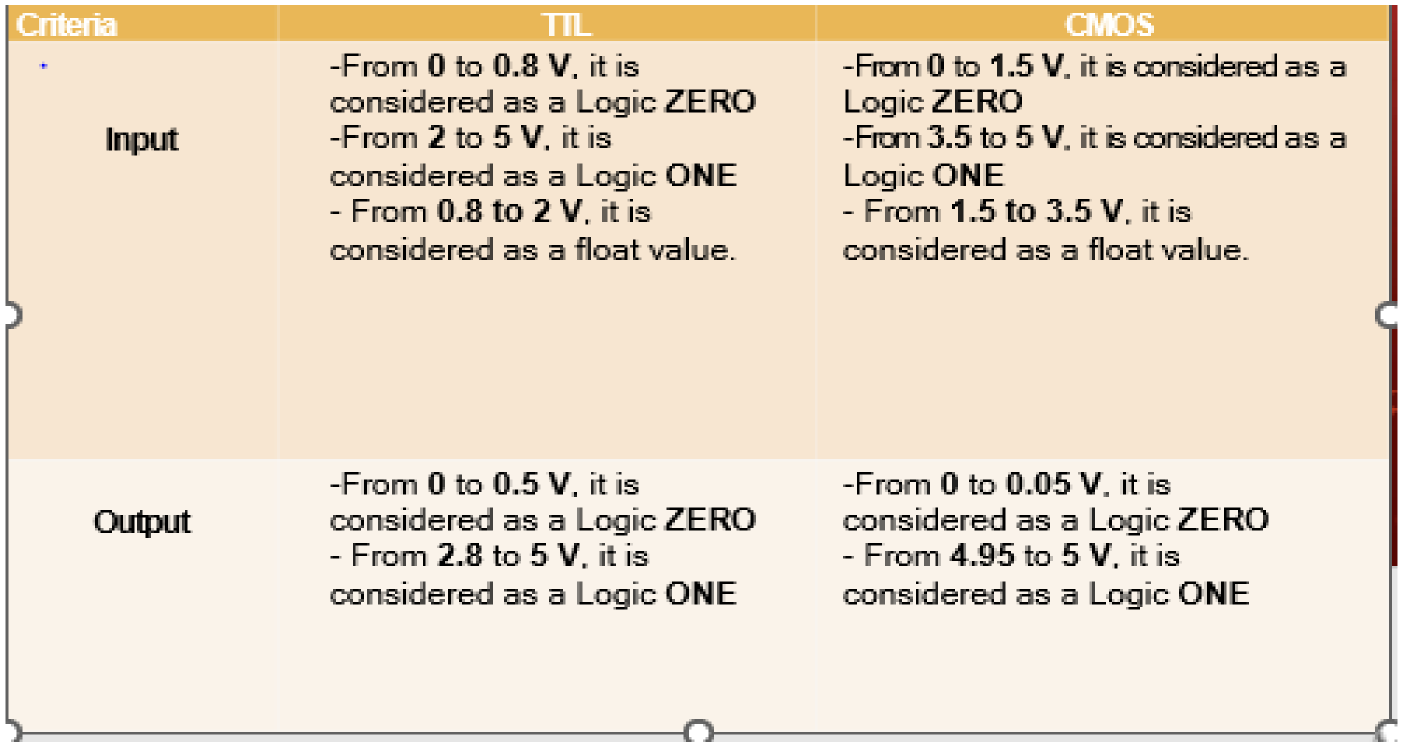
  -The level of this

  -transistor

  are different

  at logic Zero

  or logic One:



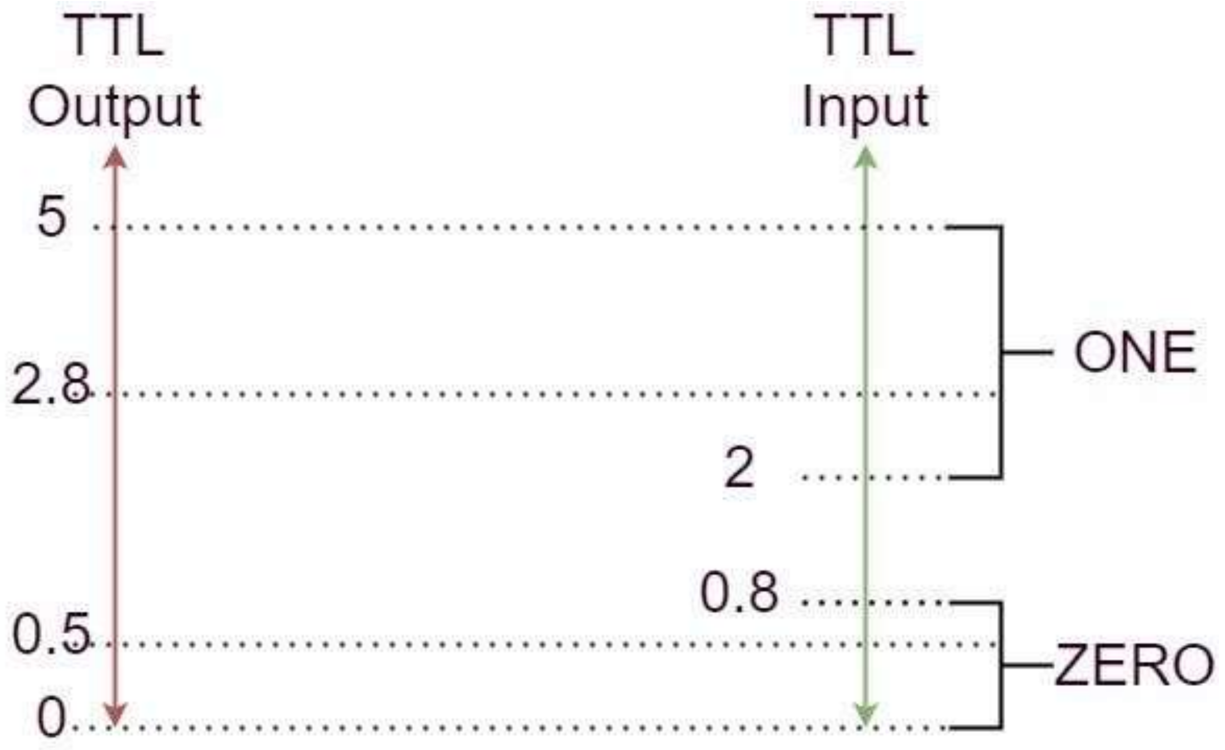
Criteria	TTL	CMOS
Input	-From 0 to 0.8 V, it is considered as a Logic ZERO -From 2 to 5 V, it is considered as a Logic ONE - From 0.8 to 2 V, it is considered as a float value.	-From 0 to 1.5 V, it is considered as a Logic ZERO -From 3.5 to 5 V, it is considered as a Logic ONE - From 1.5 to 3.5 V, it is considered as a float value.
Output	-From 0 to 0.5 V, it is considered as a Logic ZERO - From 2.8 to 5 V, it is considered as a Logic ONE	-From 0 to 0.05 V, it is considered as a Logic ZERO - From 4.95 to 5 V, it is considered as a Logic ONE

Communication Protocol:

Specs of Communication Protocol:

- Digital Level:
  - If **TTL** communicates with **TTL**:

    there is no problems because zero and one level at output is lower than their levels at input.

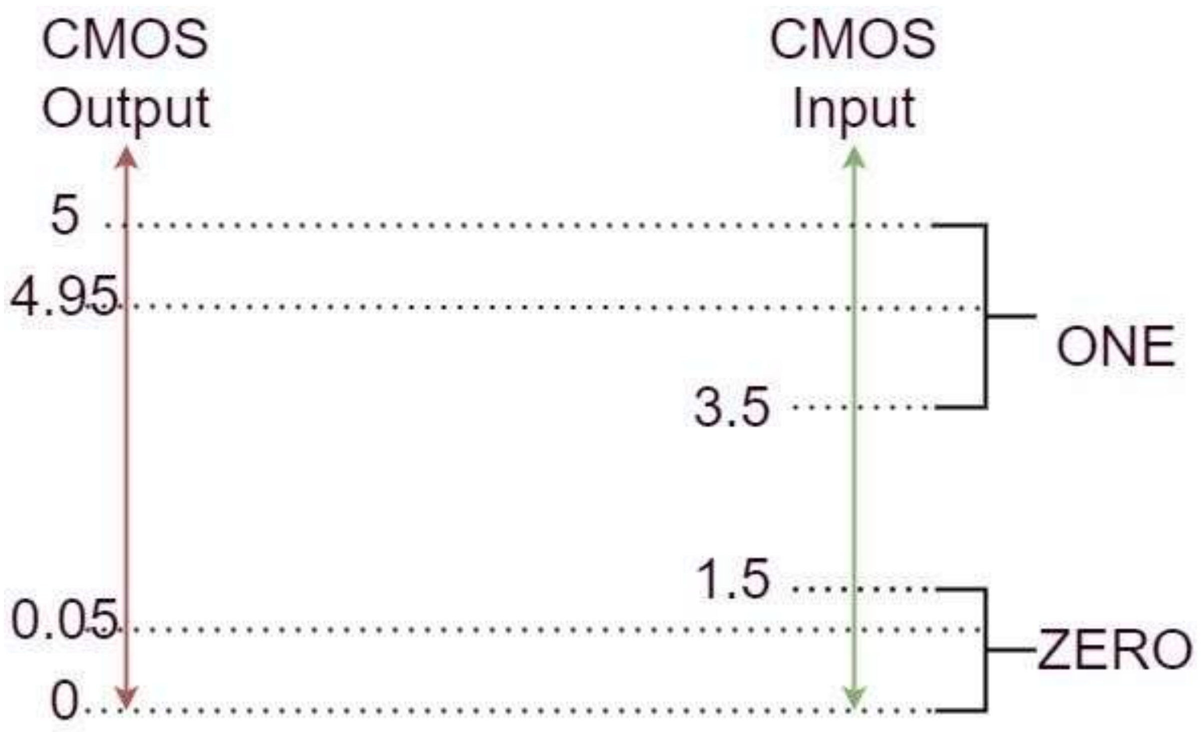


Communication Protocol:

Specs of Communication Protocol:

- Digital Level:
  - If **CMOS** communicates with **CMOS**:

    there is no problems because zero and one level at output is lower than their levels at input.

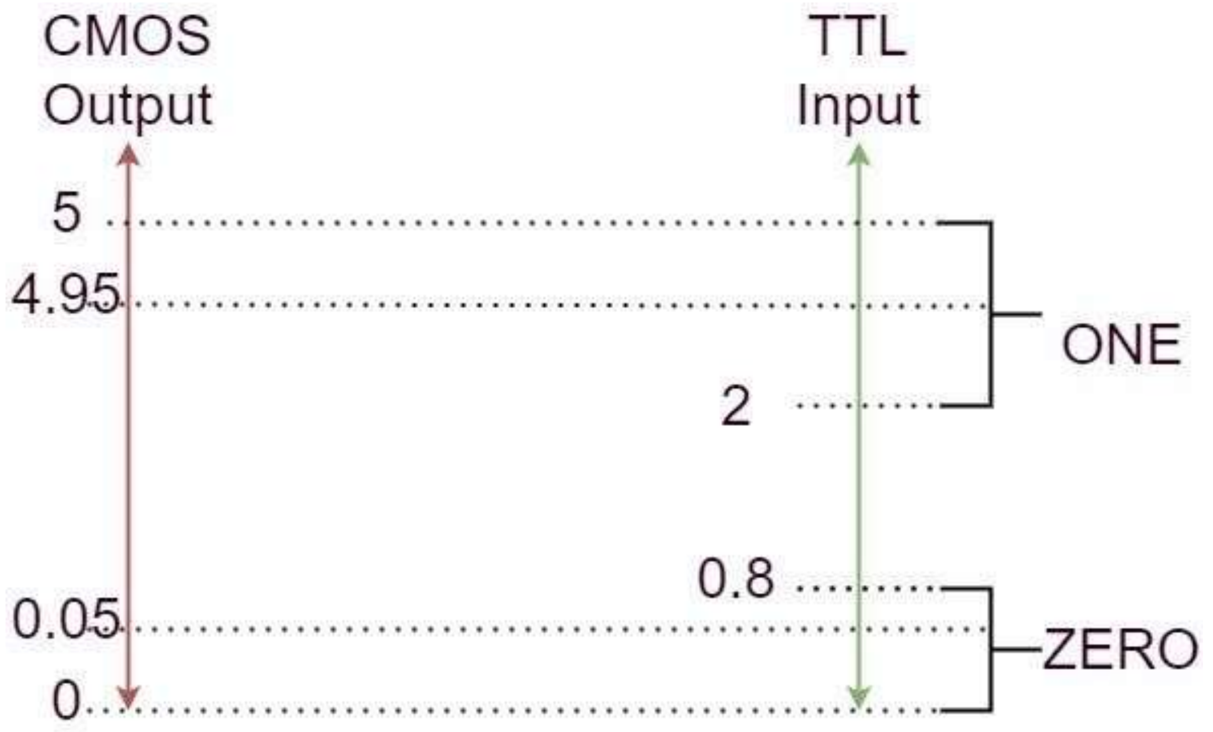


Communication Protocol:

Specs of Communication Protocol:

- Digital Level:
  - If **CMOS** communicates with **TTL**:

    there is no problems because zero and one level at output is lower than their levels at input.

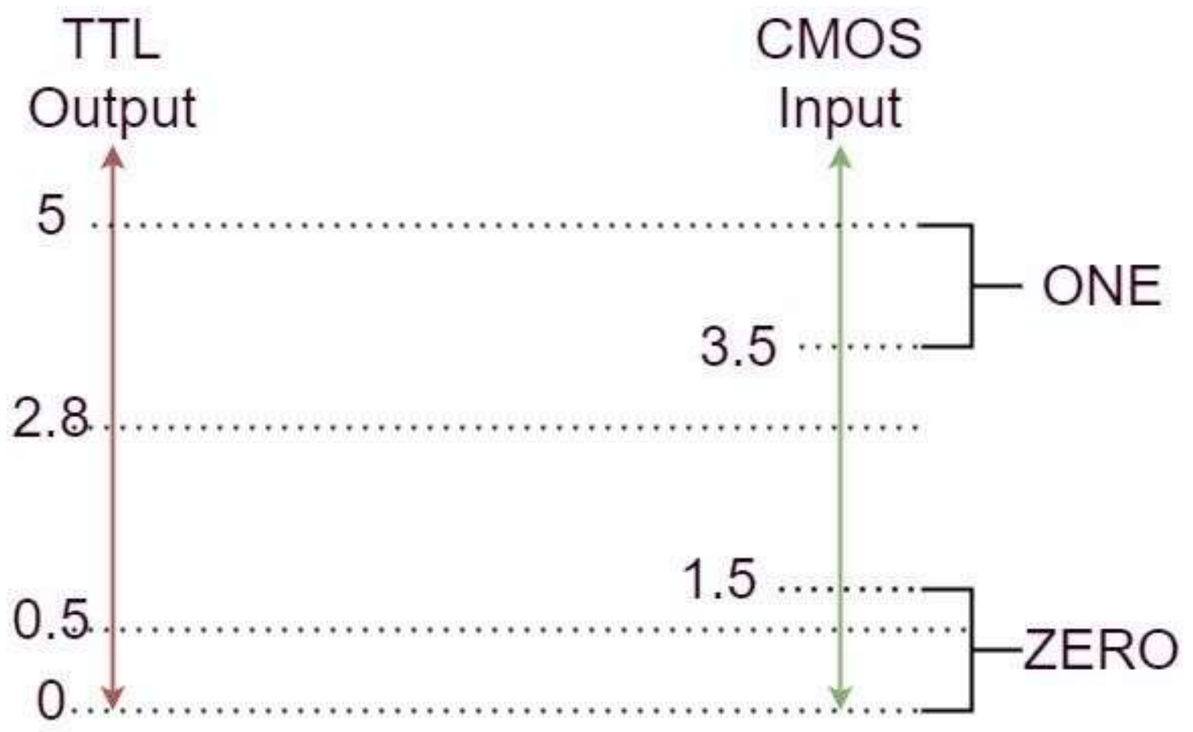


Communication Protocol:

Specs of Communication Protocol:

- Digital Level:
  - If **TTL** communicates with **CMOS**:

    there is no problems at zero logic, but one logic if TTL acts at its max error CMOS will read it as float.



Universal Asynchronies Receiver Transmitter:

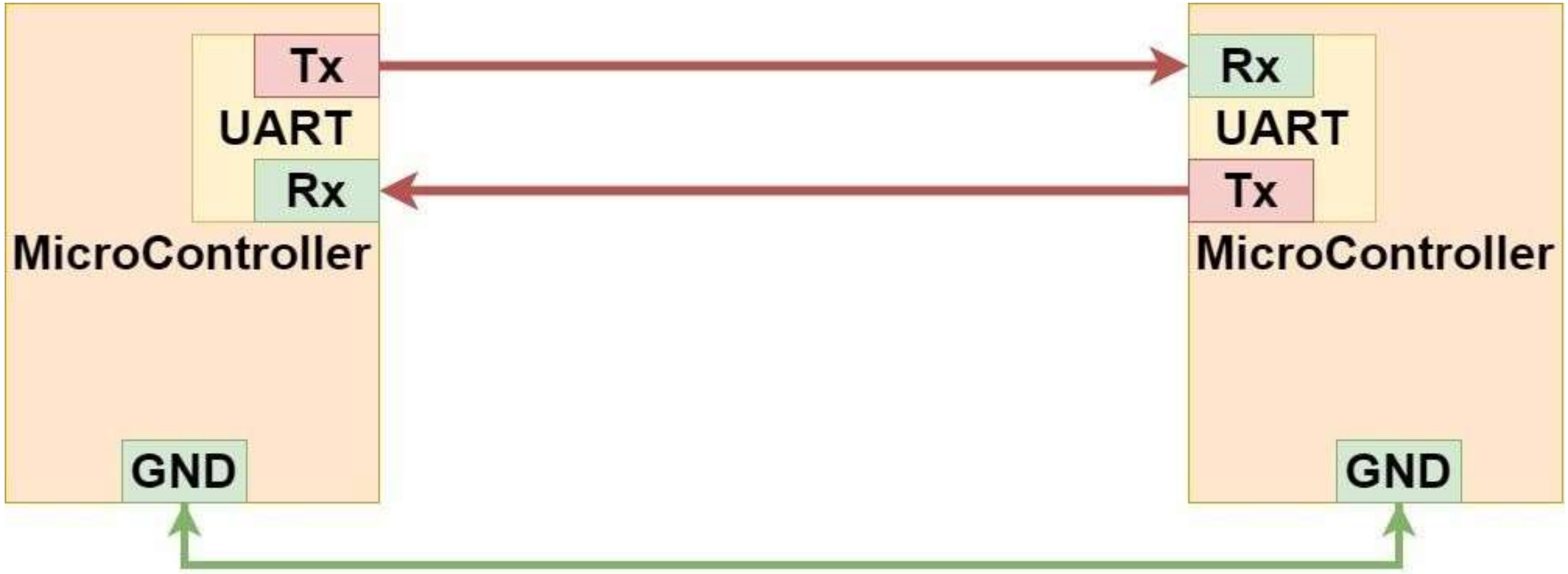
Specs of UART:

- UART is a highly flexible serial communication device.
  The main features are:
  - Wired & Serial.
  - Asynchronies.
  - Peer to Peer.
  - Half & Full Duplex.
  - Max Throughput = 81%.
  - CMOS & TTL.

Universal Asynchronies Receiver Transmitter:

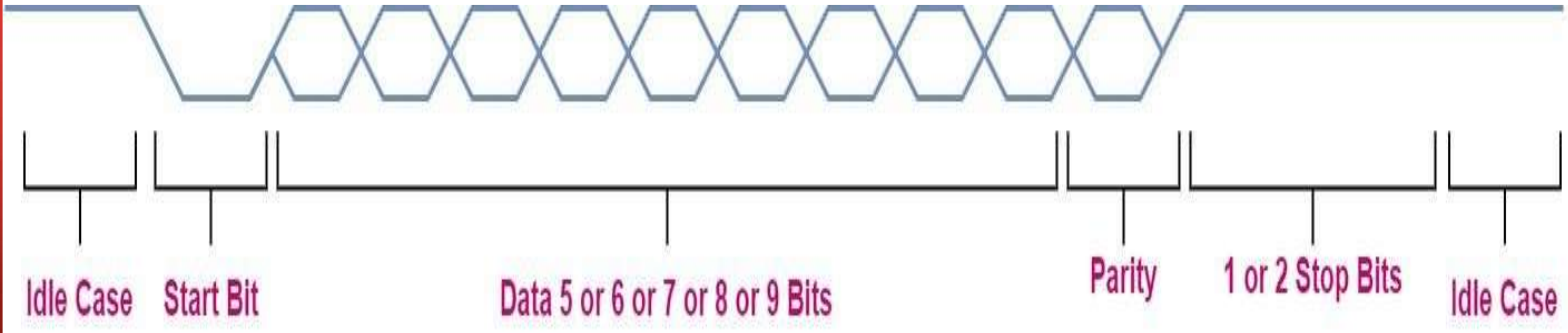
Hardware Connection of UART:

- The grounds must be unified, to become similar overall the electronic circuit, it is also applies if any two electrical components or more connected together into the same circuit.



Universal Asynchronies Receiver Transmitter:

Frame Data Format of UART:



Universal Asynchronies Receiver Transmitter:

Frame Data Format of UART:

- Idle Case:
  this is the normal case of bus when there is no data transferred.

- Start Bit:
  this is a bit which is an inverted level of idle case to manage the receiver to detect if the protocol starts.

- Data Bits:
  these are five or six or seven or eight or nine bits can be transferred per time.
  the data length is configurable using the UART registers, they will be declared later.

<u>Universal Asynchronies Receiver Transmitter:</u>

Frame Data Format of UART:

- <u>Parity Bit</u>:
  it is a kind of check mechanism, there are two kinds of parity mechanisms, it is used to check if the received data is transmitted correct or not, but how it works:
  - **<u>Even Parity</u>**:
    To understand how it works, we will take an example, like:
    assume that the transferred data is 0b00101100.
    parity mechanism counts the number of high level "ONEs" exist into the transferred data, then it assigns the parity by the value that makes the all ONEs into data and parity bit must be even, so, according to the pervious example, parity bit must be assigned by "**1**", if the data as example is 0b00101101, the parity bit in this case must be "**0**".

Universal Asynchronies Receiver Transmitter:

Frame Data Format of UART:

- Parity Bit:
  it is a kind of check mechanism, there are two kinds of parity mechanisms, it is used to check if the received data is transmitted correct or not, but how it works:
  - **Odd Parity**:
    To understand how it works, we will take an example, like:
    assume that the transferred data is 0b01101011.
    parity mechanism counts the number of high level "ONEs" exist into the transferred data, then it assigns the parity by the value that makes the all ONEs into data and parity bit must be even, so, according to the pervious example, parity bit must be assigned by "**0**", if the data as example is 0b00101101, the parity bit in this case must be "**1**".

Universal Asynchronies Receiver Transmitter:

Frame Data Format of UART:

- Parity Bit:
  the transmitter will calculate the parity of data according to its type, then the receiver will also calculate the parity of received data “it must be similar with the type of transmitter parity or the receiver will generate a parity error”, then the receiver will compare the received parity with the calculated parity, if they are not equal, the receiver also will generate a parity error, like the following example:
  - if the transmitted data is “0b01001110” and the type of parity is even, so the value of parity will be “**0**”, but data is changed due to noise like EMF and becomes “0b01101110”, so the receiver will calculate its parity “which it was configured as even as we mentioned”, the parity or receiver will be “**1**”, so a parity error will be fired. **But what will happen if two bits are changed?...........**

Universal Asynchronies Receiver Transmitter:

Frame Data Format of UART:

- Parity Bit:

  we will used the same previous example, the transmitted data is "0b01001110" and the type of parity is even, so the value of parity will be "**0**", but data is changed due to noise like EMF and becomes "0b00101110", so the receiver will calculate its parity "which it was configured as even as we mentioned", the parity or receiver will be "**0**", so no parity error will be fired.

  So according to the weakness of parity mechanism, we rarely use it for checking on data, so into our session we will disable the parity mechanism.

Universal Asynchronies Receiver Transmitter:

Frame Data Format of UART:

- Stop Bits:

  - It can be configured to be one or two bits.

  - It is used to indicate that the frame of data has been finished.

  -Its level is the inversion of start bit level, it returns the bus to it idle state again.

  - these bits or this bit must be sent every data transferred through UART, it will be declared why later.

Universal Asynchronies Receiver Transmitter:

Calculation of Throughput of UART:

- Minimal Throughput of UART:

  -To calculate the minimal throughput of UART, we must reduce The numerator of throughput and increase the denominator of throughput.

  $$\text{- Tp} = \frac{five\ bits\ of\ data}{start\ bit + five\ bits\ of\ data + parity\ bit +\ two\ stop\ bits} = 55.56\%$$

- Maximal Throughput of UART:

  - To calculate the maximal throughput of UART, we must reduce The denominator of throughput and increase the numerator of throughput.

  $$\text{- Tp} = \frac{nine\ bits\ of\ data}{start\ bit + nine\ bits\ of\ data + stop\ bit} = 81.81\%$$

AMIT

Universal Asynchronies Receiver Transmitter:

Calculation of Throughput of UART:

- Using nine bit data will made me to read or write them within two clock cycles or more, because ATMEGA32 is an eight bit data bus, so to make our code easier, we will use eight bit data in frame.

- We will use in our frame:
  - Start bit.
  - Eight bit data.
  - No parity.
  - One stop bit.

- This frame is called 8N1 frame “8 data, No parity, 1 stop bit”.

Universal Asynchronies Receiver Transmitter:

Calculation of Throughput of UART:

- 8N1 Throughput of UART:
  - from the following formula:

$$- \mathrm{Tp} = \frac{eight\ bits\ of\ data}{start\ bit + eight\ bits\ of\ data + stop\ bit} = 80\ \%$$

- This throughput is near than the maximal throughput,
  they are almost similar, but 8N1 frame is easier than the maximal because of accessing the data at more than one clock cycle.

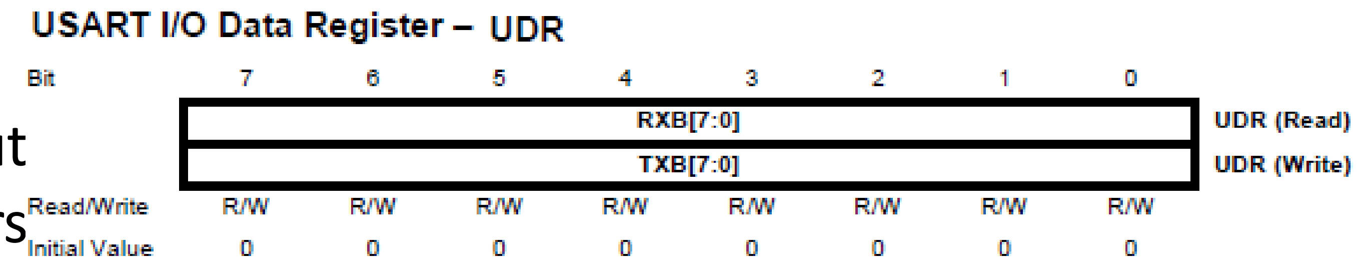
Universal Asynchronies Receiver Transmitter:

Register Description of UART:

- UART I/O Data Register "**UDR**":

  this register is used for transmitting data "if it is written" or receiving data "if it is read".

  UDR has a single location, but it is connected to two buffers that are accessed according to the kind of operation, I mean if UDR is written, the data will be written into "**TXB**" buffer, but if UDR is read, the data will be read from "**RXB**" buffer.



USART I/O Data Register – UDR

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
Read/Write	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	

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

Universal Asynchronies Receiver Transmitter:

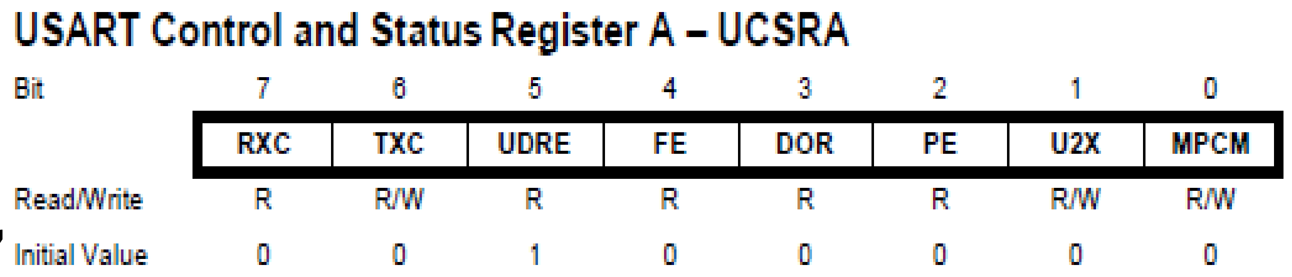
Register Description of UART:

- USART Control and Status Register A "**UCSRA**":
  - **Bit 7** – RXC: USART Receive Complete:

    This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty.

    If the receiver is disabled, the receive buffer will be flushed and consequently the RXC bit will become zero.

    The RXC Flag can be used to generate a Receive Complete interrupt.



USART Control and Status Register A – UCSRA

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

Universal Asynchronies Receiver Transmitter:

Register Description of UART:

- USART Control and Status Register A “**UCSRA**”:
  - **Bit 6** – TXC: USART Transmit Complete:
    This flag bit is set when written data has been completely transmitted.
    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.
  - **Bit 5** – UDRE: USART Data Register Empty:
    The UDRE Flag indicates if the transmit buffer (UDR) is ready to receive new data, It means that the buffer is empty, and therefore ready to be written if its value is One.
    The UDRE Flag can generate a Data Register empty Interrupt.
    UDRE is set by default at start, because its reset value is One.

Universal Asynchronies Receiver Transmitter:

Register Description of UART:

- USART Control and Status Register A “**UCSRA**”:
  - **Bit 4** – FE: Frame Error:

    This is used to detect if the next character in the receive buffer had a Frame Error when received, 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.

Universal Asynchronies Receiver Transmitter:

Register Description of UART:

- USART Control and Status Register A "**UCSRA**":
  - **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), it is a new character waiting in the receive Shift Register, 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.

Universal Asynchronies Receiver Transmitter:

Register Description of UART:

- USART Control and Status Register A “**UCSRA**”:
  - **Bit 2** – PE: Parity Error:
    We explained before how the parity bit can check if the data is changed.
    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.

Universal Asynchronies Receiver Transmitter:

Register Description of UART:

- USART Control and Status Register A "**UCSRA**":
    - **Bit 1** – U2X: Double the USART 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 "we will explain later when these number are used to be a divisor" effectively doubling the transfer rate for asynchronous communication.

Universal Asynchronies Receiver Transmitter:

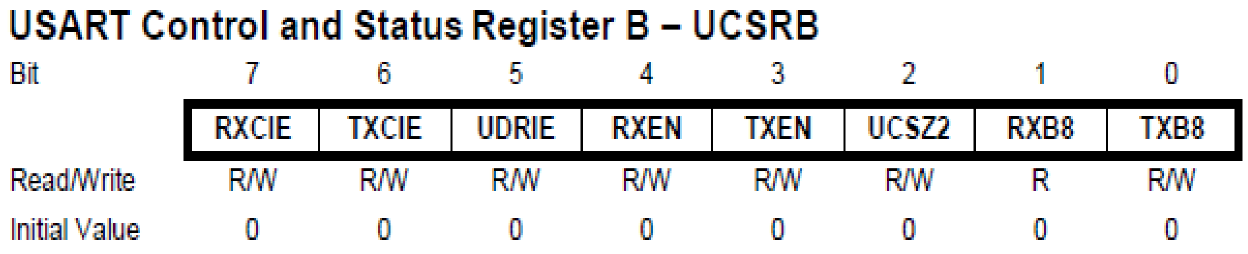
Register Description of UART:

- USART Control and Status Register A “**UCSRA**”:
  - **Bit 0** – MPCM: Multi-processor Communication Mode:
    Tis bit will convert the UART from asynchronization to synchronization mode and from peer to peer relation to Masters/Slaves mode.
    In this mode, nine bit data must be used, because the ninth bit’s value will indicate if the frame is address frame or data frame For more detailed information see “Multi-processor Communication Mode” on page 155 in the datasheet.
    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.

Universal Asynchronies Receiver Transmitter:

Register Description of UART:

- USART Control and Status Register B “**UCSRB**”:
  - **Bit 7** – RXCIE: RX Complete Interrupt Enable:
    This is the PIE bit of reception interrupt, the interrupt will be fired if RXC is raised and GIE is set.



USART Control and Status Register B – UCSRB

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

  - **Bit 6** – TXCIE: TX Complete Interrupt Enable:
    This is the PIE bit of transmission interrupt, the interrupt will be fired if TXC is raised and GIE is set.
  - **Bit 5** – UDRIE: USART Data Register Empty Interrupt Enable
    This is the PIE bit of UDR Empty interrupt, the interrupt will be fired if UDRE is raised and GIE is set.

Universal Asynchronies Receiver Transmitter:

Register Description of UART:

- USART Control and Status Register B "**UCSRB**":
  - **Bit 4** – RXEN: Receiver Enable:
    Writing this bit to one enables the USART Receiver Circuit.
    Disabling the Receiver will flush the receive buffer invalidating the FE, DOR, and PE Flags.
  - **Bit 3** – TXEN: Transmitter Enable:
    Writing this bit to one enables the USART Transmitter Circuit.
    The disabling of the Transmitter (writing TXEN to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the transmit Shift Register and transmit Buffer Register do not contain data to be transmitted.

Universal Asynchronies Receiver Transmitter:

Register Description of UART:

- USART Control and Status Register B "**UCSRB**":
  - **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:
    RXB8 is the ninth data bit of the received character when operating with serial frames with nine data bits.
    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.
    Must be written before writing the low bits to UDR.

Universal Asynchronies Receiver Transmitter:

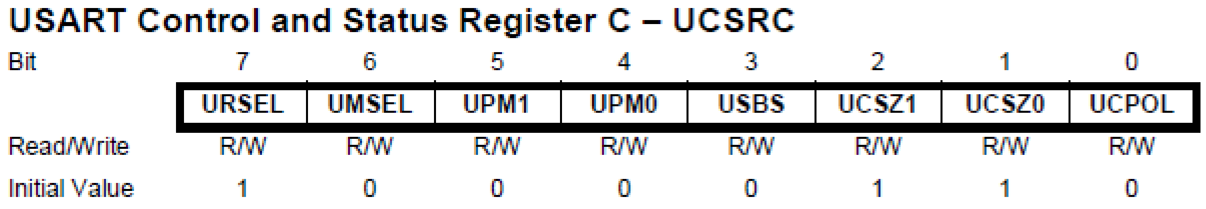
Register Description of UART:

- USART Control and Status Register C “**UCSRC**”:

  The UCSRC Register shares the same I/O location as the UBRRH Register, so to select the UCSRC or UBRRH, see the description of Bit7 “URSEL”.

  - **Bit 7** – URSEL: 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.



USART Control and Status Register C – UCSRC

Bit	7	6	5	4	3	2	1	0
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	1	0	0	0	0	1	1	0

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

    This bit selects between Asynchronous “written to zer0” and Synchronous “written to one” mode of operation.

Universal Asynchronies Receiver Transmitter:

Register Description of UART:

- USART Control and Status Register C “**UCSRC**”:
  - Bit 5:4 – UPM1:0: Parity Mode:

    These bits enable and set type of parity generation and check. If enabled, the transmitter will automatically generate and send the parity of the transmitted data bits within each frame.

    The Receiver will generate a parity value for the incoming data and compare it to the UPM0 setting.

    If a mismatch is detected, the PE Flag in UCSRA will be set.

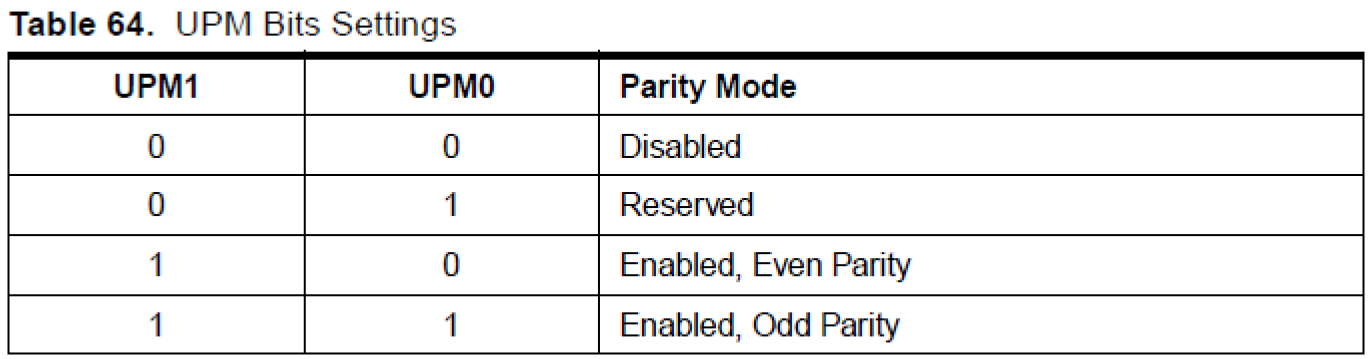


Table 64. UPM Bits Settings

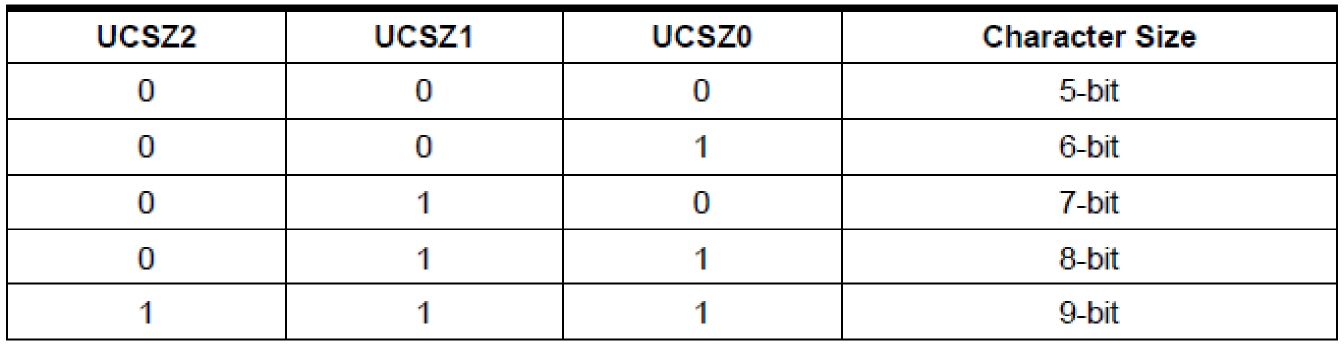
UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Universal Asynchronies Receiver Transmitter:

Register Description of UART:

- USART Control and Status Register C “**UCSRC**”:
  - Bit 3 – USBS: Stop Bit Select:
    This bit selects the number of Stop Bits to be inserted by the Transmitter “One stop bit, if written to ZERO, and Two stop bit, if written to ONE ”. The Receiver ignores this setting.
  - Bit 2:1 – UCSZ1:0: Character Size:
    The UCSZ1:0 bits combined with the UCSZ2 bit in UCSRB sets the number of data bit (Character Size) in a frame the Receiver and Transmitter use.

Table 66. UCSZ Bits Settings



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	1	1	9-bit

Universal Asynchronies Receiver Transmitter:

Register Description of UART:

- USART Control and Status Register C “**UCSRC**”:
    - Bit 0 – UCPOL: Clock Polarity:
      This bit is used for Synchronous mode only.
      Write this bit to zero when Asynchronous mode is used.

Universal Asynchronies Receiver Transmitter:

Register Description of UART:

- USART Baud Rate Registers “**UBRRL and UBRRH**”:

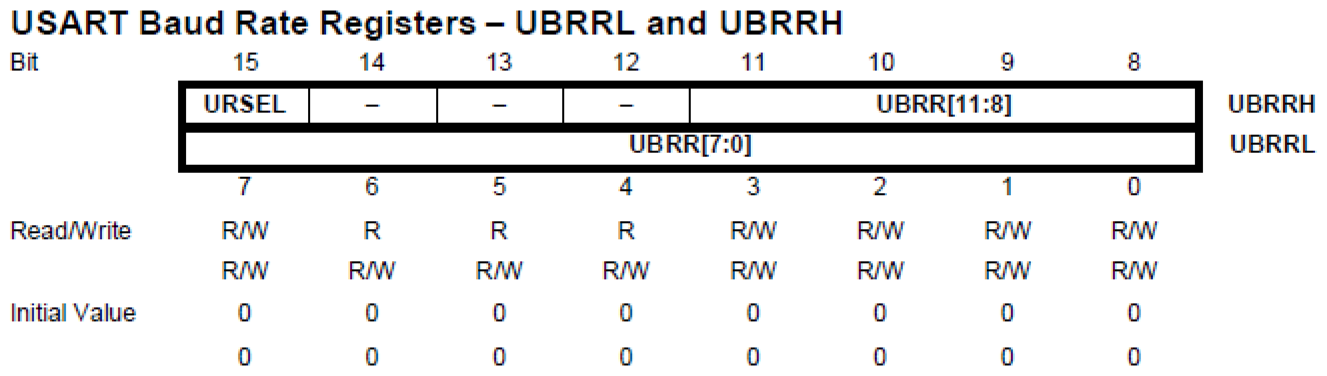
  The UCSRC Register shares the same I/O location as the UBRRH Register, so to select the UCSRC or UBRRH, see the description of Bit7 “URSEL”.

  - Bit 15 – URSEL: Register Select

    This bit selects between accessing the UCSRC or the UBRRH Register.

    It is read as zero when reading UBRRH.

    The URSEL must be zero when writing the UBRRH.



USART Baud Rate Registers – UBRRL and UBRRH

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	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	

  - Bit 14:12 – Reserved Bits:

    These bits are reserved for future use.

    These bits must be written to zero when UBRRH is written.

Universal Asynchronies Receiver Transmitter:

Register Description of UART:

- USART Baud Rate Registers "**UBRRL and UBRRH**":
    - Bit 11:0 – UBRR11:0: USART Baud Rate Register:
      This is a 12-bit register which contains the USART baud rate.

      The UBRRH contains the four most significant bits, and the UBRRL contains the 8 least significant bits of the USART baud rate.

      Ongoing transmissions by the transmitter and receiver will be corrupted if the baud rate is changed. Writing UBRRL will trigger an immediate update of the baud rate prescaler.

Universal Asynchronies Receiver Transmitter:

Accessing of UCSRC and UBRRH:

- The UCSRC Register shares the same I/O location as the UBRRH Register, so to select the UCSRC or UBRRH, see the description of Bit7 "URSEL".
- It is preferred to access these registers as a byte access by direct assignation into them.
- They are available for bit access, but take care that the reset or default value of "URSEL" is "1", so if their shared location is accessed in the beginning of program, the register that will be accessed is "UCSRC" .

```
/* Set UBRRH to  val */
 UBRRH = val;
/* Set the USBS and the UCSZ1 bit to one.*/
 UCSRC = (1<<URSEL) | (1<<USBS) | (1<<UCSZ1);
```

Universal Asynchronies Receiver Transmitter:

Accessing of UCSRC and UBRRH:

- The read access is controlled by a timed sequence.
- Reading the I/O location once returns the UBRRH Register contents. If the register location was read in previous system clock cycle, reading the register in the current clock cycle will return the UCSRC contents.
- Note that the timed sequence for reading the UCSRC is an atomic operation. Interrupts must therefore be controlled (e.g., by disabling interrupts globally) during the read operation.

```
/* Read UCSRC */
   ucsrc  =  UBRRH;
   ucsrc = UCSRC;
```

Universal Asynchronies Receiver Transmitter:

Calculation of Baud Rate of UART:

- To calculate the baud rate or UBRR value at normal speed mode, we must use the following formula:

$$\text{Baudrate} = \frac{frequency\, of\ system}{16 \times (UBRR + 1)}$$

$$UBRR = \frac{frequency\, of\ system}{16 \times\ Baudrate} - 1$$

- To calculate the baud rate at normal speed mode, we must use the following formula:

$$\text{Baudrate} = \frac{frequency\, of\ system}{8 \times (UBRR + 1)}$$

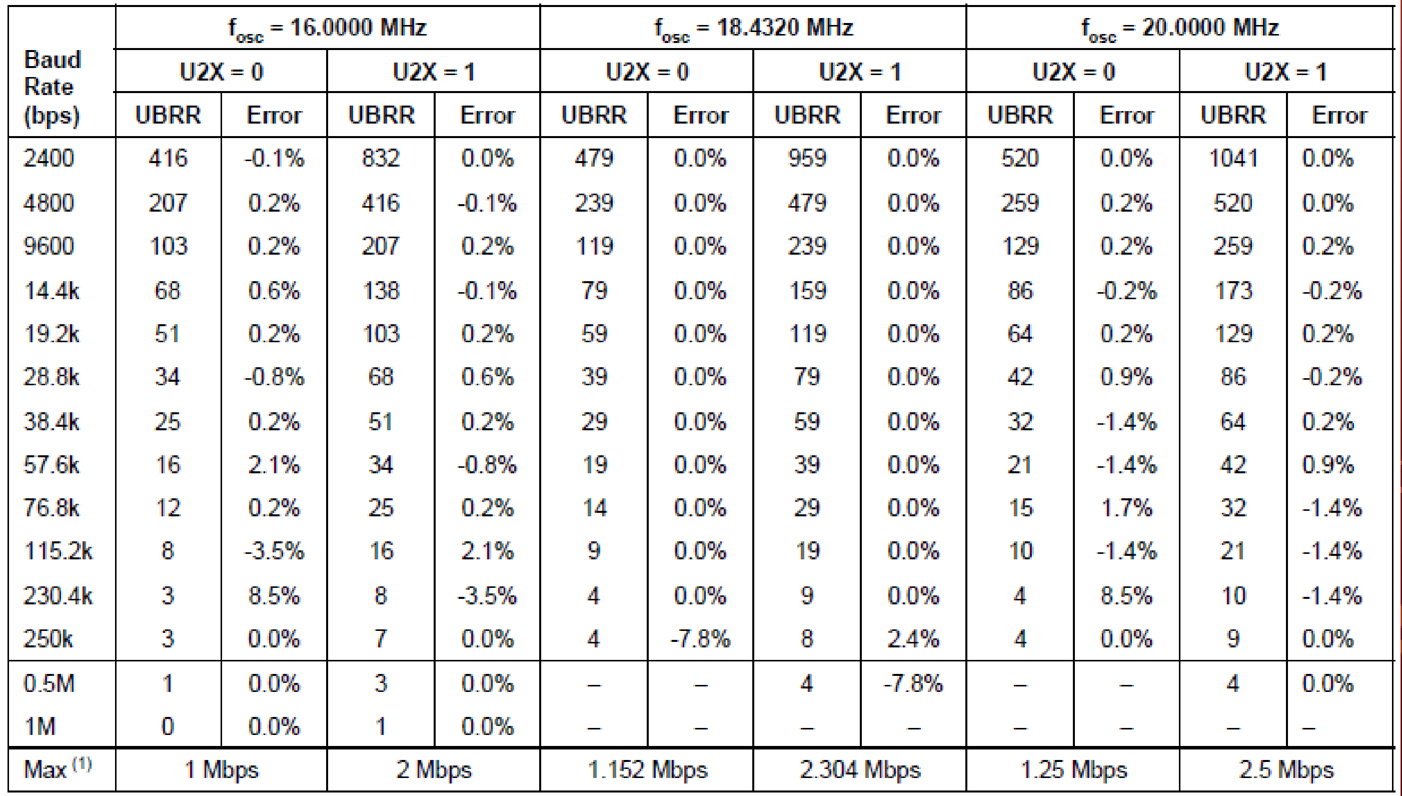
$$UBRR = \frac{frequency\, of\ system}{8 \times\ Baudrate} - 1$$

Universal Asynchronies Receiver Transmitter:

Baud Rate Error Calculation of UART:

- There are tabled exist into the data sheet that provides the UBRR values that have been calculated before according to the value of system frequency and baud rate.
- There is a column into the table, exists in page 163, includes
  the calculated error
  of baud rate, like:

Table 71. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)



Baud Rate (bps)	f_{osc} = 16.0000 MHz				f_{osc} = 18.4320 MHz				f_{osc} = 20.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	–	–	4	-7.8%	–	–	4	0.0%
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
Max [(1)]	1 Mbps		2 Mbps		1.152 Mbps		2.304 Mbps		1.25 Mbps		2.5 Mbps	

Universal Asynchronies Receiver Transmitter:

Baud Rate Error Calculation of UART:

- As example, we will take the value of 9600 baud rate, normal mode, 16 MHz, so the value of UBRR will be:

$$UBRR = \frac{16{,}000{,}000}{16 \times 9600} - 1 = 103.166667$$

so, UBRR will be updates only by 103 because it is an integer register, so if we calculate what the output Baud Rate exactly at 103:

$$\text{Baudrate} = \frac{16{,}000{,}000}{16 \times (103+1)} = 9615\ bit\backslash s$$

so the error rate will be:

$$\text{Error Rate} = \frac{calculated\ baudrate - desired\ baudrate}{desired\ baudrate} \times 100$$

$$= \frac{9615 - 9600}{9600} = +0.16\ \%$$

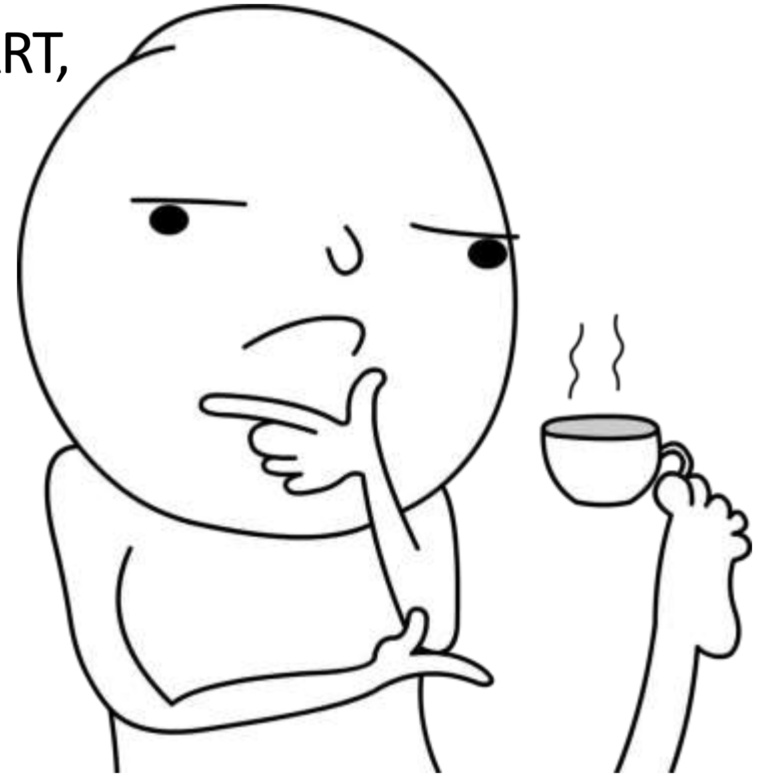
Universal Asynchronies Receiver Transmitter:

Baud Rate Error Calculation of UART:

- So, what will be happened if our microcontroller communicate with another one that has as example a negative error "- 0.6 %", so its real baud rate is as example "9578",
  so if these micros need to communicate using UART,
  will they understand themselves?
  As we mentioned, the expected answer is
  No way,
  but in real life they can understand
  themselves normally,

But

HOW??!!

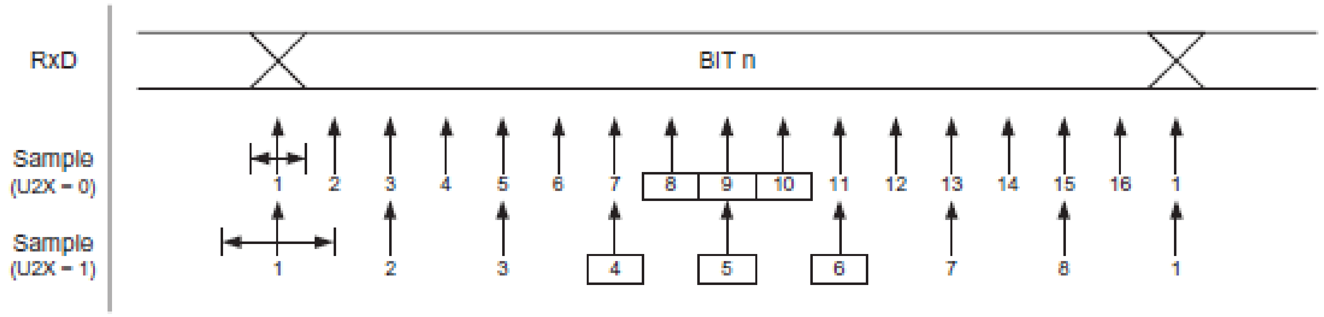


Universal Asynchronies Receiver Transmitter:

Reception Mechanism of UART:

- When the receiver clock is synchronized to the start bit, the data recovery can begin.
- The data recovery unit uses a state machine that has 16 states for each bit in normal mode and 8 states for each bit in Double Speed mode.

Figure 74. Sampling of Data and Parity Bit



- Figure 74 shows the sampling of the data bits and the parity bit. Each of the samples is given a number that is equal to the state of the recovery unit. 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. The center samples are emphasized on the figure by having the sample number inside boxes.

Universal Asynchronies Receiver Transmitter:

Reception Mechanism of UART:

- The majority voting process is done as follows: If two or all three samples have high levels, the received bit is registered to be a logic 1. If two or all three samples have low levels, the received bit is registered to be a logic 0.
- This majority voting process acts as a low pass filter for the incoming signal on the RxD pin.
- The recovery process is then repeated until a complete frame is received. Including the first stop bit.
- Note that the receiver only uses the first stop bit of a frame.

Universal Asynchronies Receiver Transmitter:

Baud Rate Error Calculation of UART:

- Remember that the division factor into the baud rate equation is “16” at normal mode, “8” at high speed mode, this factor is used because the speed of receiver circuit is faster than the transmitter by 16 times or 8 times, but why??!!

  The receiver circuit samples only one bit within 16 clock cycles or 8 clock cycles , it samples the bit three times at the 8th, 9th and 10th clock, or 4th, 5th and 6th, and it will store the average of them.

- So if there is a small difference to speeds of UARTs, so if the transmitter is faster than the receiver, the reading clocks will be shifted to high, so it may be at 11th, 12th and 13th as example, so if it is read within 16 or 8 clock cycles, it will be sampled correctly.

Universal Asynchronies Receiver Transmitter:

Baud Rate Error Calculation of UART:

- This declares that the stop bit must be written every transmitted data, because if the stop bit is written, the receiver circuit will be turned off, so the delayed of shifted cycles will be cleared until the next data, the start bit will wake the receiver circuit up again with no shifting into the clock.

- The conclusion is that: if there is a small difference from the speeds of UARTs, it can be ignored because the UART using majority win mechanism and the reception circuit works faster than transmission circuit by 16 times, so it can bear the high or low shift, and the stop bit is a mandatory every transmitted data to reset the reception circuit.

Universal Asynchronies Receiver Transmitter:

Initialization of UART:

- First, select your
  - data length.
  - parity type.
  - number of stop bits.
  - the value of baud rate.
  - type of UART (receiver/transmitter or both).
  - type of synchronization (synch. Oy asynch.) .
  - Type of event trigger (polling or interrupt).
  - Speed mode (Normal or Double).

Universal Asynchronies Receiver Transmitter:

Initialization of UART:

- Simple initialization function:

```
void UART_init(void)
{
        /* normal speed, peer to peer*/
        UCSRA = 0;
              /* Enable receiver and transmitter, polling*/
        UCSRB = (1<<RXEN)|(1<<TXEN);
        /* Set frame format: 8data, no parity, one stop bit, asynch */
        UCSRC = (1<<URSEL)|(3<<UCSZ0);
         /* Set baud rate 9600 at 16MHz*/
        UBRRH = 0;
        UBRRL = 103;
}
```

Universal Asynchronies Receiver Transmitter:

Sending Data of UART:

- Simple transmission function “using UDRE flag”:

```
void UART_sendData(u8 data)
{
        /* Wait for empty transmit buffer */
        while ( !( UCSRA & (1<<UDRE)) );


        /* Put data into buffer, sends the data */
        UDR = data;
}
```

Universal Asynchronies Receiver Transmitter:

Sending Data of UART:

- Simple transmission function "using TXC flag": void UART_sendData(u8 data)

```
{
    /* Put data into buffer, sends the data */
    UDR = data;


    /* Wait for transmission complete */
    while ( !( UCSRA & (1<<TXC)) );
}
```

Universal Asynchronies Receiver Transmitter:

Receiving Data of UART:

- Simple transmission function “using TXC flag”: u8 UART_receiveData(void)

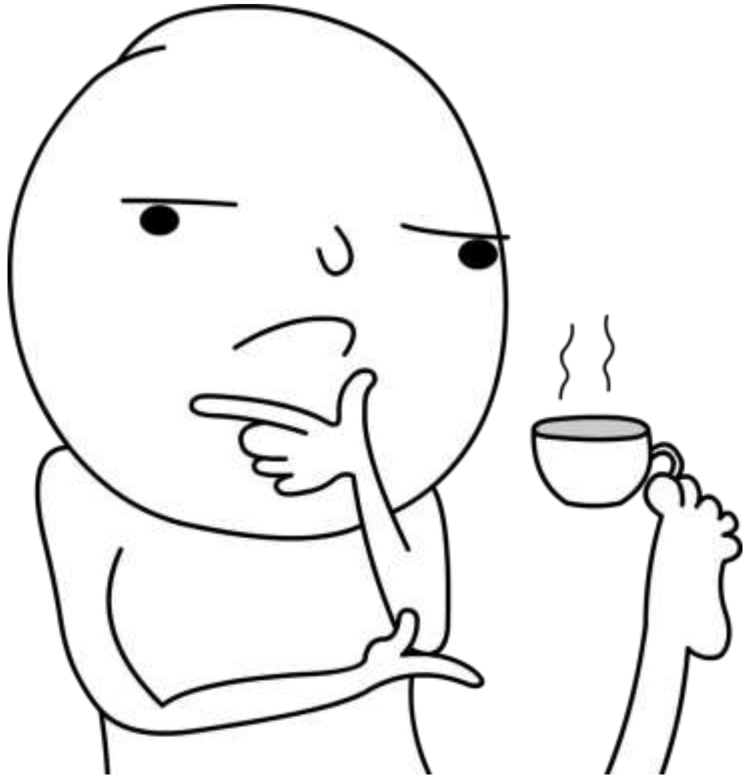
```
{
    /* Wait for data to be received */
    while ( !(UCSRA & (1<<RXC)) );


    /* Get and return received data from buffer */
    return UDR;
}
```

Universal Asynchronies Receiver Transmitter:

Initialization of UART:

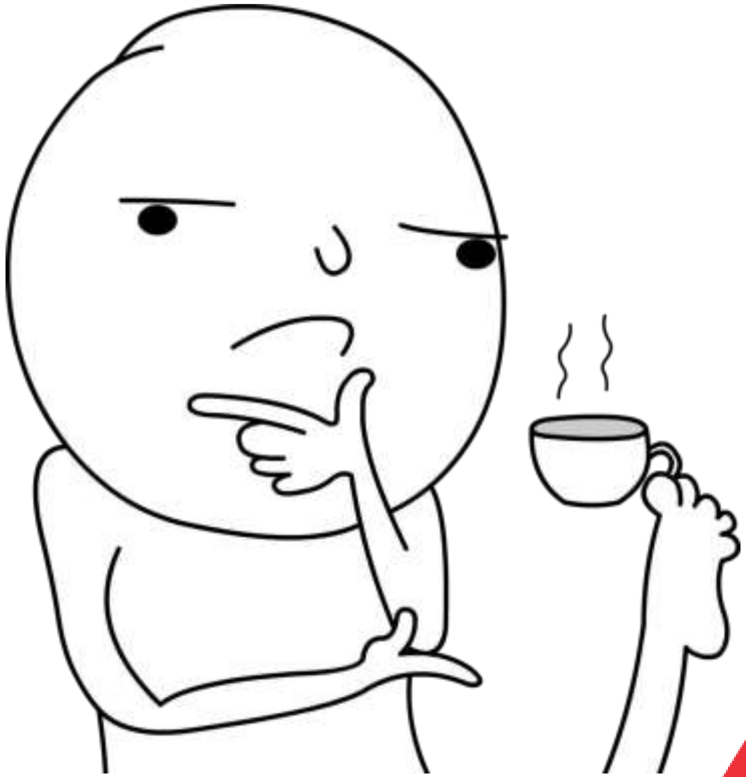
Now,
think well to create two functions:
sendString?
receiveString?



UART Driver:

Time To

Code



THANK YOU!