6

UART Port Based Serial Data Communication

6.1 Architecture of Hardware UART Port

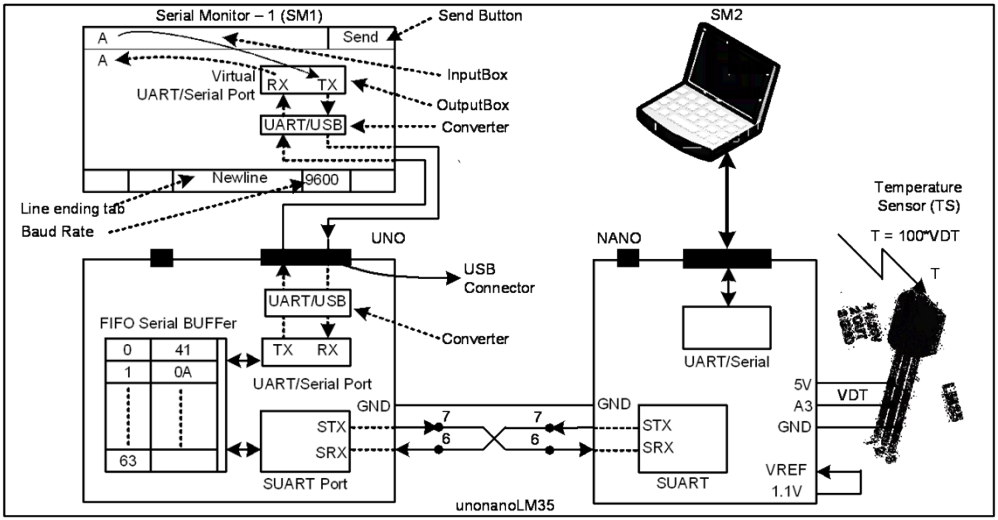


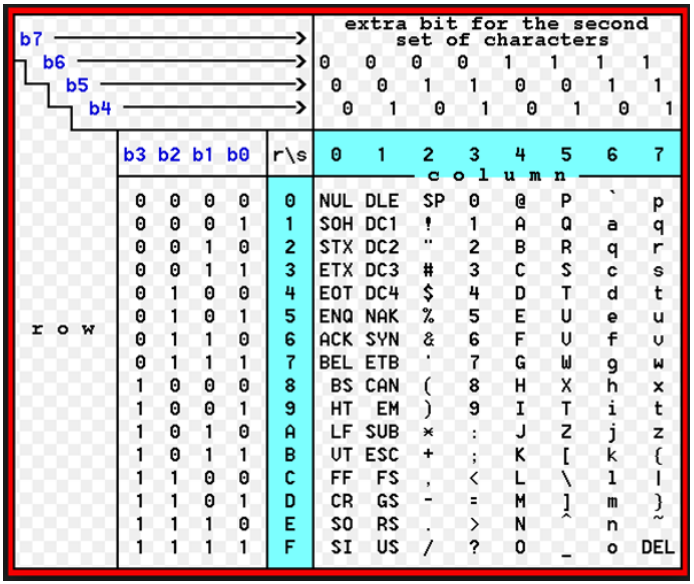
Figure-6.1: *UART and SUART Ports of ATmega328P MCU*

(1) The ATmega328P MCU contains only one UART/Serial Port (Fig-6.1) to exchange data "bit-by-bit" with Serial Monitor or any other controller or sensor. The UART Port stands for "Universal Asynchronous Reception and Transmission" Port.

(2) A SUART Port (Software UART Port, Fig-6.1) can also be created by including the following codes in the sketch.

```
#including<SoftwareSerial.h>       //we need a Library in include in the IDE and sketch
SoftwareSerial  SUART(6, 7);       //DPin-6 = SRX line; DPin-7 = STX line
SUART.begin(9600);                 //SUART Port is enabled at 9600 Bd. Frame length = 10
```

(3) If we enter character **A** in InputBox of Serial Monitor (Fig-6.1) and then click on Send button, then this 8-bit data 01000001(0x41 in hex base) called ASCII Code (Fig-6.2) is transferred to UNO over TX (transmit line). The TX line of SM is connected with RX (receive line) of UNO. The lower 7-bit of "01000001" is actually the ASCII Code (Fig-6.2). The upper bit (MSBit) is always 0. The whole 8-bit is called "character code". ASCII stands for American Standard Code for Information Interchange. When we say one character, we refer to these letters: A – Z, a – z, 0 – 9, punctuation marks, and control characters. There are some characters like **control codes**, which are not printable characters.



b7	b6	b5	b4	b3	b2	b1	b0	r\s	0	1	2	3	4	5	6	7
									extra bit for the second set of characters							
0	0	0	0	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	0	0	1	0	2	STX	DC2	"	2	B	R	q	r
0	0	1	1	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	1	0	0	1	9	HT	EM	)	9	I	T	i	t
1	0	1	0	1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
1	0	1	1	1	0	1	1	B	VT	ESC	+	;	K	[	k	{
1	1	0	0	1	1	0	0	C	FF	FS	,	<	L	\	l	|
1	1	0	1	1	1	0	1	D	CR	GS	-	=	M	]	m	}
1	1	1	0	1	1	1	0	E	SO	RS	.	>	N	^	n	~
1	1	1	1	1	1	1	1	F	SI	US	/	?	O	_	o	DEL

row — column

Figure-6.2: *ASCII Codes Table*

(4) Before the data byte 01000001 of Step-3 begins to move towards UNO, a START-bit (LOW) goes first (Fig-6.3); then goes the LSBit of character code; then goes the remaining 7-bit of character code; then goes a STOP-bit (HIGH). Parity bit is an optional bit. Now, there are in total 10 bits, and it is called **asynchronous frame** (async frame) (Fig-6.3). The frame length is 10-bit.

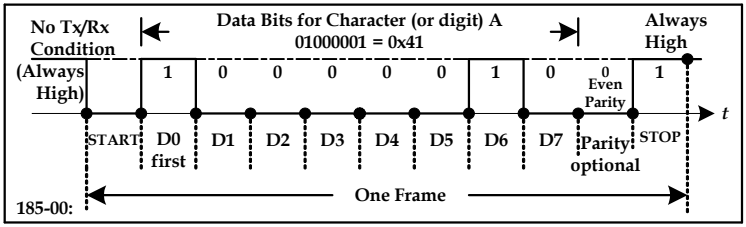


Figure-6.3: *Asynchronous frame for character A*

(5) Baud Rate (Bd) refers to number of bits being transferred and received in one second time. The UART Port of Fig-6.1 is enabled at 9600 Bd by executing the following code: A frame will take about 41 µs time to arrive from SM to UNO.

```
Serial.begin(9600); //1-START Bit, 8-Charcater Code, No-Parity, 1-STOP Bit, Frame = 10 bit
```

(6) If "Line ending tab" in the SM of Fig-6.1 is selected to "Newline" option, then the code 0x0A (in C-Language it is '\n') will be transmitted after the transmission of all the characters of the InputBox. In ASCII Table of Fig-6.2, the Newline is shown as LF (Line Feed). It is a non-printable character. This Newline character is usually used to mark the end of the string (sequence of characters) being sent from the InputBox.

(7) We may execute one of the following codes to send Newline character from UNO to the OutputBox of Serial Monitor.

```
Serial.print('\n'); //'\n' is replaced by 0x0A for ASCII code of Newline character
⇒ Serial.println():    //same as above
⇒ Serial.write(0x0A);  //This line is called frame. It puts the bit pattern on UART Port.
```

(8) The Line ending tab of the Serial Monitor (SM) has the following options:

i. **No line ending —** When this option is selected, the SM does not send any code to the UNO after sending the ASCII codes for the characters of the InputBox.

ii. **Newline (NL) —** When this option is selected, the SM sends 0x0A to the UNO after sending the characters of the InputBox. In C Language, the character representation of 'Newline' is '\n'. In Arduino, it is: Serial.println().

iii. **Carriage return (CR)** — When this option is selected, the SM sends 0x0D to the UNO after sending the characters of the InputBox. In C Language, representation is '\r'.

iv. **Both NL & CR (NLCR)**— When this option is selected, the SM sends 0x0D and then 0x0A to the UNO after sending the characters of the InputBox. In C Language, the code is: "\r\n").

(9) When the data frame of Fig-6.3 arrives at UNO, the START and STOP bits are taken out. The remaining 8-bit data is saved at Location-0 of the FIFO (first-in first-out) type Seral BUFFer of the UART Port of MCU. The BUFFer can hold 64 bytes data (ASCII or pure binary).

(10) If we enter A123 in the InputBox of SM and then click on Send button with Newline —

i. How many data frame will be transferred by the SM? Ans: 5

ii. What bit pattern would be saved at Location-0 of the BUFFer? Ans: 0010001 (0x41)

iii. What bit pattern would be saved at Location-1 of the BUFFer? Ans: 00110001 (0x31)

(11) Assume that we have sent 123 and NL (Newline) from the InputBox of SM. As a result, 0x31 0x32 0x33 0x0A have arrived to UNO and get stored into the FIFO (first-in first-out) Serial Buffer. Write commands:

i. To check that at least one character has arrived in the buffer.

```
byte n = Serial.available();  //n == number of characters present in buffer
if(n == 0)
{
     Serial.println("No character is present in the buffer);
}
Serial.println("At least one character is in the buffer); //max == 4
```

ii. Assume that all 4 characters have arrived and get stored in the buffer. Write commands to read the characters from buffer and save them into variables.

```
char x1 = Serial.read();          //x1 = ? ASCII code of 1 = 0x31
char x2 = Serial.read();          //x2 = ? ASCII code of 2 = 0x32
char x3 = Serial.read();          //x3 = ? ASCII code of 3 = 0x33
char x4 = Serial.read();          //x4 =  ?   ASCII of NL = 0x0A
```

iii. Re-write codes of Step-*ii* using *for()* loop to reduce number of lines.

```
char myData[4];
for(int i=0; i<4; i++)
{
     myData[i] = Serial.read();
}
```

iv. Write codes to read arrived characters including NL from buffer and save them in an array.

```
char myArray[4]; //array size = string length + 1 ; last location will hold null-byte (0)
int i = 0;
char myArray[4];
int i = 0;

void loop()
{
     byte n = Serial.available();
     if(n != 0)
     {
          char x = Serial.read();
          myArray{i] = x;
          i++;
     }
}
```

v. Re-write codes of Step-*iv* so that all arrived characters are saved in the myArray[] except NL.

```
char myArray[12];
void loop()
{
     byte n = Serial.available();
     if(n != 0)
     {
               char x = Serial.read();
               if (x != '\n')  //save if '\n' == NL == 0x0A is not detected
               {
                    myArray[i] = x;
                    i++;    //0 1 2   i =3
               }
               else
               {
                    myArray[i] = '\0';//insert null-byte at end to print a char type array
                    Serial.print(myArray) ; //shows: 123
                    //To print a char type array, you must insert null-byte (0x00 = '\0')
                    //as the last element of the array in order to stop printing
                    memset(myArry, 0x00, 12);  //all memory locations are cleared
          }
     }
}
```

vi. Re-write Step-*v* with minimum codes:

```
char myArray[12];

byte n = Serial.available();
if(n != 0)
{
     //read and save except NL == '\n' from buffer until NL ('\n') is detected
      byte m = Serial.readBytesUntil('\n', myArray, 12);
     //next lines are executed if '\n' is found or timeout occurs or 12 chars are received
      myArray[m] = '\0';    //insert null-byte at location pointed by m
      Serial.println(myArray); //shows: 123
      //-------------------------------------------------
       memset(myArray, 0x00, 12);     //reset all locations of array to 0s
}
```

(12) Let us note down that the MCU always executes *Serial.write()* method to put "8-bit character code = the binary bits of ASCII code" onto the UART Port. Thus, when the *Serial.print('A')* code is executed, it is broken down as follows:

```
Serial.print('A');   //opening and closing single quotes are used for one charcater
⇒ Serial.write(0x41); //'A' is replaced by its ASCII code; write() method is called frame
```

(13) The MCU executes the following code to send/show multiple characters onto the OutputBox of SM. Note that a pair double quote (" and ") is used for a string (multiple) of characters.

```
Serial.print("A12");     //three frames will be sent one after another
//---------------------------------
⇒ Serial.print('A');    //we write this code
⇒ Serial.write(0x41)    //MCU executes this code
//---------------------------------
⇒ Serial.print('1');
⇒ Serial.write(0x31)
//---------------------------------
⇒ Serial.print('2');
⇒ Serial.write(0x32);
```

(14) What will appear on the OutputBox of SM after the execution of the following instructions?

```
(1)  Serial.print(12, DEC); //DEC = base 10; HEX = base 16, BIN = base 2; SM shows: 12
     ⇒ Serial.print('1');
     ⇒ Serial.write(0x31);  //1 will appear on SM as 0x31 is the ASCII code of 1
     //------------------
     ⇒ Serial.print('2');
     ⇒ Serial.write(0x32);  //2 will appear on SM as 0x32 is the ASCII code of 2
     //----------
(2)  Serial.print(0x34, DEC);      //SM shows: 52
     //0x34 will be converted to 52 (16x3 + 1x4) decimal as the base is DEC = 10.
     ⇒ Serial.print(52, DEC);
     ⇒ Serial.print('5');
     ⇒ Serial.write(0x35);  //5 will appear on SM
```

```
(3) Serial.print(0xC4, HEX);      //SM shows: C4
    //0xC4 will remain as it is (in hex form) as the base is hex (16).
    ⇒ Serial.print('C');
    ⇒ Serial.write(0x43);  //C will appear on SM
(4) Serial.print(0xC5);      //when base is not given, the base is 10
(5) Serial.print("0xC5");    //SM shows: 0xC5 as it is a string
(6) Serial.print(123.768, 2);//shows: 123.76
    //arg1=float number; arg2=digits to show after decimal point
(7) Serial.println();//inserts one line gap (non printable character; called NewLine = NL
    ⇒ Serial.print('\n');
    ⇒ serial.write(0x0A);
(8) Serial.println("AB"); // 3 frames would be sent: 0x41 for A, 0x42 for B, x0A for ln
    Serial.print("AUST"); //
(9) Serial.print("123");
(10) char myArray[] = {'A', '1', '2', '3', '\0'};     //\0 is called NULL-byte
    ⇒ char myArray[] = {0x41, 0x31, 0x32, 0x33, 0x00}
    ⇒ char myArray[] = "A123";     //null-byte is automatically inserted
    ⇒ Serial.print(myArray);  //A123; to print string, there must be null-byte at end
```

(15) You are asked to read a data byte from the serial BUFFer of Fig-6.1. Which option of the following will you choose?

i. You just perform a read operation on the BUFFer.

ii. You will first check that there is at least one data item in the FIFO Serial BUFFer (Fig-6.1) and then you will perform read operation.

a. The following instruction is executed to check that the BUFFer contains at least one data item.

```
byte n = Serial.available();   // n = 0 means no data item is present in BUFFer
if(n != 0 )    //there is character (s) in serial BUFFer
{
   char y = Serial.read();    //bring data item from Location-0 of BUFFER into y
}
```

b. The following instruction is executed to send back the value of above **y** into OutputBox of Serial Monitor.

```
Serial.print(y);
```

(18) Assume that the string "AUST" is sent from SM1 of Fig-6.1 with Newline option. Write code for UNO to receive all the characters until Newline character is found. Send back the received string to the OutputBox of SM1. A string/cstring refers to a sequence of characters. To print a string, we must insert null-byte (0x00 = '\0') as a last element in the array.

```
char recvArray[10];      //receiver array to store the received characters
int i = 0;               //array index

void loop()
{
     byte n = Serial.available();   /check that BUFFER has data
     if(n != 0)    //BUFFEer contains at least one data item
```

```
    {
         char y = Serial.read();     //read the arrived character
         if(y == 0x0A)  //Newline character has arrived; no more character
         {
              recvArry[i] = 0x00;   //null-byte goes into location recvArray[4]
              Serial.print(recArry);  /shows: AUST
              while(1);    //wait for ever
         }
         else
         {
              recvArray[i] = y;       //Newline has not arrived; read character and save
              i++;               //increase array index; i = 4
         }
    }
}
```

(19) Repeat Step-18 using *Serial.readBytesUntil()* methd.

```
char recvArray[10];

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  byte n = Serial.available(); //n = 1
  if ( n != 0)  //Buffer contains at least one data byte
  {
     //keep receiving data bytes until Newline is found
     byte m = Serial.readBytesUntil('\n', recvArray, 10);//arg3 = array size
     // m = number of charcaters stored in Buffer except Newline
     recvArray[m] = '0'; //insert null-byte
     Serial.println(recvArray);
  }
}
```

The *readBytesUntil()* method terminates whenever one of the following events occur:

'\n' is received or
"number of declared characters are received" or
"timeout (default 1sec)".

(20) Assume that serial BUFFer of UNO of Fig-6.1contains: ABCDE; where, A has entered as the last element. Write code bring them out from the serial BUFFer, show them on OutputBox of SM1, and store them in an array named *char myData[].*

```
char myData[6];          //array size is 6 to hold 5 items and null-byte
byte i = 0;              //array index/pointer
while(Serial.available() != 0)   //Serial.available() = 0 after reading all 5 items
```

```
{
     myData[i] = Serial.read();   //ASCII codes are coming
     Serial.print(myData[i[))
     i++;
}
```

6.2 Programming of the UART Port

(1) Write sketch so that when you send message "**Ignite L**" from SM of Fig-6.4 with Newline option, then **L** of UNO will be ON. The message **L is ON** will also appear on OutputBox of SM.

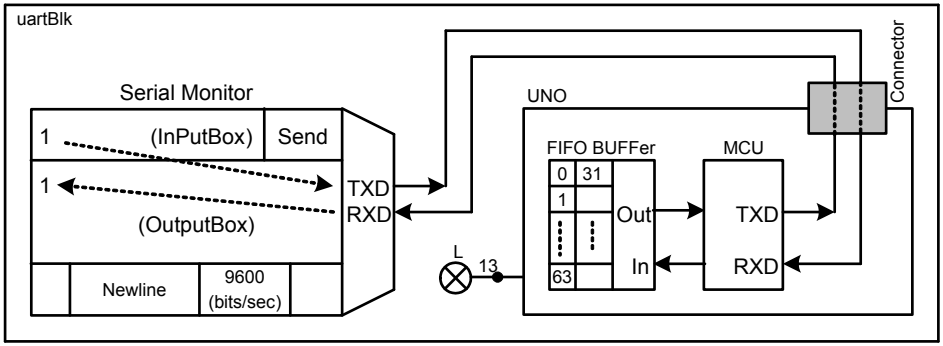


Figure-6.4:

```
char recString[10]; //array to store received characters
int i = 0;          //array index

void setup()
{
  Serial.begin(9600);
  pinMode(13, OUTPUT);
}

void loop()
{
  byte n = Serial.available();  //check that a character has arrived
  if(n !=0)                     //a character has arrived
  {
    char y = Serial.read();     //put the ASCII of character in y
    if(y == '\n')  //C code for Newline character is found;
    {
      recString[i] = '\0';                    //insert null-byte
      bool m = strcmp(recString, "Ignite L"); //comparing two arrays
      if(m == 0x00)               //strng matches
      {
        digitalWrite(13, HIGH);   // L is ON
        Serial.print("L is ON");  //message on Serial Monitor
        while(1);                 //wait here
      }
    }
    else
    {
```

```
      recString[i] = y; //Newline character has not arrived; keep saving characters
      i++;
    }
  }
}
```

(2) [strtoul *String To Unsigned long function***]** Write codes for UNO of Fig-6.4 to receive comma separated three decimal numbers (like: 1234, 567, 89) from InputBox of SM (with Newline option) and then extract the individual numbers using **strtoul()** (string to unsigned long integer) function and then save them into three variables.

Working Principle of *strtoul() Function***:**
The function parses (reads a character from a character type array and checks if it is a decimal or hexadecimal digit) a string (array of characters) and stops at the non-digit character. The ASCII codes of the digits which have been parsed so far are converted into a numerical number and are saved in a variable. For example:

```
char myArray[] = "1234, 567, 89";
char *temPtr;     //pointer variable
unsigned int x1 = strtoul(myArray, &temPtr, 10);//arg3(base-100 says string has decimal digits
Serial.println(x1, DEC);  //shows: 1234
```

The arg3 (= 10) says that the function looks for decimal digits in the array. The parsing stops at the first comma (, = non-digit character); there remains the sub-array/sub-string (, 567, 89) which will be parsed next time to extract 567 and 89. Now, we need a pointer variable (temPtr) to point the beginning of the sub-array. The arg2 (&temPtr) passes "the address of the character at which parsing had stopped" to the pointer and now the *strtoul()* function can locate the beginning of the sub-array.

i. Enter 1234, 567, and 89 in the InputBox of SM and then select "Newline" and then click on the Send button. As a result the following ASCII codes will be travelling towards UNO: 31 32 33 34 2C 20 35 36 37 2C 20 38 39 0A. (The spaces are not here but are shown for clarity.)

ii. Codes for UNO to receive the above frames and then extract the decimal numbers using **strtoul** (string to unsigned long) function and then save them into variables ***x1***, ***x2***, and ***x3***.

```
 Char myArray[20];  //array to hold arrived characters
Char *temPtr;       //pointer to hold the beginning address of the remaining sub-string

Void loop()
{
     while(Serial.available() > 0)   //check that the buffer holds at least one character
     {
          byte m = Serial.readBytesUntil('\n', myArray, 20); //save characters except NL

          myArray[m] = '\0';  //null byte
          Serial.println(myArray);   //shows: 1234, 567, 89
          //---------------------------------------------
```

```
        unsigned int x1 = strtoul(myArray, &temPtr, 10);
        Serial.println(x1, DEC);      //shows: 1234
        Serial.println(x1, HEX);      //shows: x1 = 0x02D4

        //Next parsing will begin by skipping the comma (,). Space will be auto skipped
        unsigned int x2 = strtoul(temPtr+1, &temPtr, 10);     //x2 = 0x0237 = 567
        Serial.println(x2, DEC);      //shows: 567

        byte x3 = strtoul(temPtr, &temPtr, 10);     //x3 = 0x59 = 89 decimal
        Serial.println(x3, DEC);      //shows: 89
    }
}
```

(3) [atoi() *ASCII to Integr function***]** The *atoi()* function parses the data bytes if a character array for decimal digits and stops when a non-decimal character is found. The parsed characters are converted into decimal umber and are saved in a variable. The function does not require null-byte at the end of the array.

(4) [atof() *ASCII To Float* function**]** The *atof()* function receives characters for the symbols of a float number (say: 123.65) from SM1 of Fig-6.1 and then extract/save the float number. . The function does not require null-byte at the end of the array.

```
char myData[10];

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  byte n = Serial.available(); //n = 1
  if ( n != 0)  //Buffer contains at least one data byte
  {
     //keep receiving data bytes until Newline is found
     byte m = Serial.readBytesUntil('\n', myData, 10);//arg3 = array size
     // m = number of charcaters stored in Buffer except Newline
     float x = atof(myData);
     Serial.println(x, 2);
  }
}
```

6.3 Problems and Solutions

1 Write sketch to parse and save the following five hexadecimal numbers (NL as terminating character) arriving from the InputBox of Serial Monitor. Use *for()* loop wherever you can to reduce the number of code lines.: 12F34, A45, 78, 1C, 36.

2 **[***application of* **SUAR Port] In** Fig-6.1, NANO acquires temperature signal from LM35 sensor at 2-sec interval, show it on SM2, and save in *float myTemp* variable. Write sketches for both UNO and NANO so that UNO collects temp signal from NANO and shows it on SM1.