Serial Communication

- Why serial data communication

- différent types of serial data transfer

- Key features of 8051 serial interface

- RS-232 Compatible Serial Port (UART)
    - Advantage of integrated serial UART port
    - SFR's for serial communication
    - Support of Polling and Interrupt

- How to program SCON SFR to configure serial interface in different modes of operation
- Setting the Serial Port Modes

- PCON – Msb 7 - serial baud rate modify bit

- Setting the Serial Port Baud Rate
    - Calculation of TH1 values for various baud rates

Pseudo code
- Writing a byte to the serial port
- Reading a byte from the serial port

- **Hands on**
    - Transfer letter 'A' serially at 4800 baud rate continuously using 8b data and 1 stop and start bits
    - Transfer message "YES' serially at 9600 baud rate continuously, 8b data, 1 stop bit and start bit
    - Echo back the user input using serial port (use polling) (Receive and transmit back)
    - Using printf() from standard io libraries
    - Writing ISR for serial interrupt

- **Why serial data communication**
    - Cost
    - Noise
    - Availability of suitable media
    - Inherent device characteristics
    - Small connector
    - Distributed computing
    - sensor networks

- **different types of serial data transfer**
    - simplex
    - half duplex
    - full duplex

- **Key features of 8051 serial interface**
    - Full duplex
    - Receive buffered
    - SFR – SBUF – 2 buffers for tx and rx
    - 4 different modes – different apps and data rate and no of bits 8/9
    - Wide baud rate range support – 12MHZ OSC - 1Mbps to 110 Mbps – high and low speed communication support
    - SFR for programming – SCON – operating modes and control functions
    - PCON – Msb to control the baud rate in serial communication and other power control bits

- **How to program SCON SFR to configure serial interface in different modes of operation**
    - Mode 0 – shift register 8b – 12MHz – 1Mbps – higher rate
    - Mode 1 – 8b UART – normal usage – variable and programmable baud rate
    - Mode 2 and 3 – 9b UART – multi processor communication
        - Higher baud rates (divide by 32 or 64) or variable

    - SM2 – enable serial device in multi processor communication mode
    - REN – enable or disable receiving of data
    - TB8
        - 0-7 is the normal 8b data
        - So now the 8th bit (ie 9 bit) can be decided by the programmer
        - It can be parity bit
        - Or it can used for multiprocessor communication – for communication in master slave mode
            - ie bit 8 which is stored in TB8 is transmitted as the 8th bit in 9b data communication in serial mode

    - RB8
        - 8 th bit received in 9b serial mode that goes into this RB8 bit
    - TI
        - used for interrupting the processor after the completion of tx of 1 byte of data
        - Automatically set by hw after tx of a byte of data
        - Can be reset by programming or whenever the processor jumpts to ISR of serial interrupt for TI

    - RI
        - for rx of 1 byte data
        - Used in the receive mode of data tx whenever we are receiving the data
        - After 1 byte of data is received or 9b of data in other modes of operation – so after 7th or 8th bit is received – hw sets this bit and it can interrupt the processor
        - It can be reset from programming or whenever the processor jumps to IRQ of serial interrupt then this RI bit is resetted automatically

- **PCON – bit 7 – msb – serial baud rate modify bit**
    - When 1 – baud rate is modified – doubled
    - When 0 – baud rate is as programmed

Serial Communication is a form of I/O in which the bits of a byte being transferred appear one after other in a timed sequence on a single wire.

Serial Communication uses two methods, asynchronous and synchronous.

The Synchronous method transfers a block of data at a time, while the asynchronous method transfers a single byte at a time.

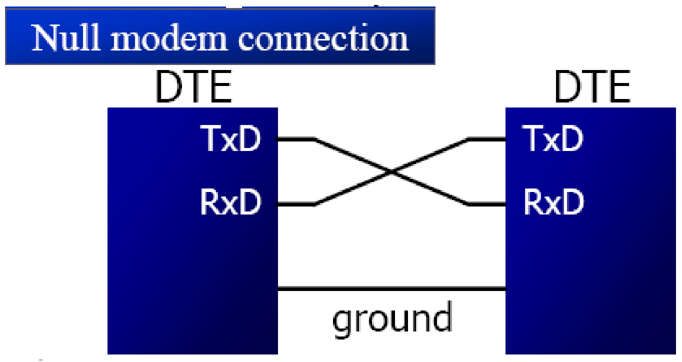
In Synchronous Communication the data get transferred based on a common clock signal.

But in Asynchronous communication, in addition to the data bit, one start bit and one stop bit is added. These start and stop bits are the parity bits to identify the data present between the start and stop bits.

The 8051 has two pins that are used specifically for transferring and receiving data serially. These two pins are called TXD and RXD and are part of the Port-3 group (Port-3.0 and Port-3.1). Pin 11 of the 8051 is assigned to TXD and pin 10 is designated as RXD. These pins are TTL compatible; therefore they require a line driver to make them RS232 compatible. The line driver chip is MAX232. The MAX232 uses +5v power source, which is same as the source voltage for 8051.

The 8051 transfers and receives data serially at different baud rates. The baud rate of the 8051 is programmed into the timers.

Generally Null Modem Connections are used for Data transfer between two devices serially.



Serial Port (UART)

- The 8051 includes a standard RS-232 compatible serial port you may use with your application programs.
- One of the 8051s many powerful features is its integrated *UART*, otherwise known as a serial port.

Advantage of integrated Serial Port

- The fact that the 8051 has an integrated serial port means that you may very easily read and write values to the serial port.
- If it were not for the integrated serial port, writing a byte to a serial line would be a rather tedious process requiring turning on and off one of the I/O lines in rapid succession to properly "clock out" each individual bit, including start bits, stop bits, and parity bits.
- However, we do not have to do this. Instead, we simply need to configure the serial ports operation mode and baud rate.
- Once configured, all we have to do is write to an SFR to write a value to the serial port or read the same SFR to read a value from the serial port.
- The 8051 will automatically let us know when it has finished sending the character we wrote and will also let us know whenever it has received a byte so that we can process it.
- We do not have to worry about transmission at the bit level--which saves us quite a bit of coding and processing time.

SFR's for Serial Port

- The 8051 uses port pins P3.1 and P3.0 for transmit and receive respectively.
- There are several SFRs that you must properly configure before the serial port will function.
- The serial interface is configured with the SFR registers SBUF, SCON and PCON.
- In addition, you must also configure Timer 1 or Timer 2 as the baud rate generator.

Support for Polling or ISR

- The 8051 provides full interrupt control for the serial port transmits and receive operations.
- You may poll the serial port control registers to determine when a character has been received (Or)
- When the next character may be sent or you may create interrupt routines to handle these operations

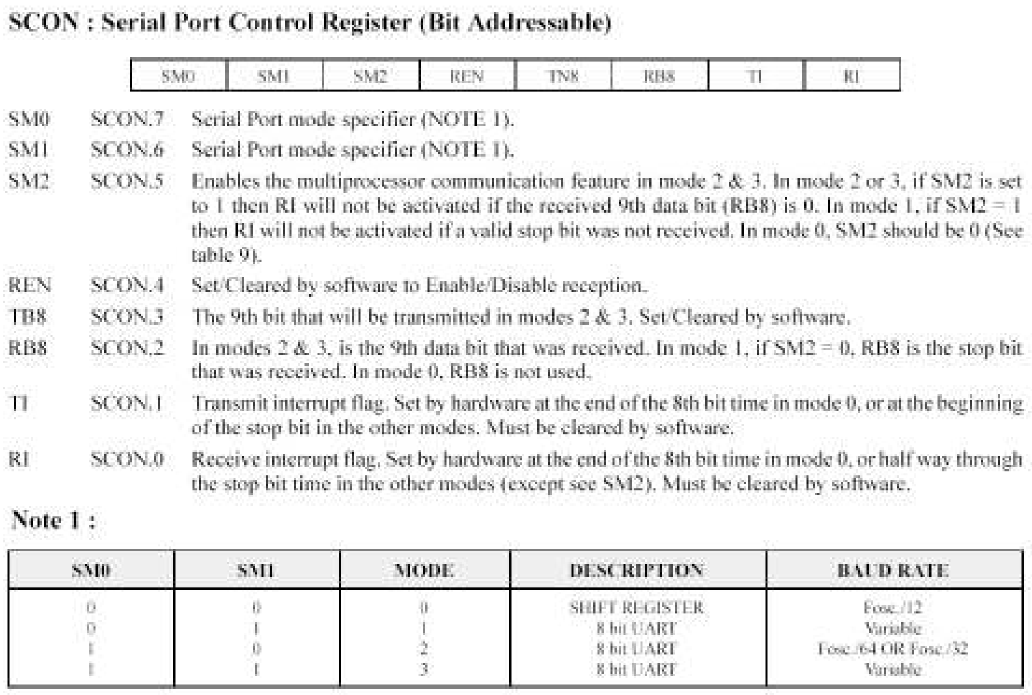
Setting the Serial Port Mode

- The first things we must do when using the 8051s integrated serial port is, obviously, configure it.

- This lets us tell the 8051 how many data bits we want, the baud rate we will be using, and how the baud rate will be determined.

Serial Control SFR (SCON)

SCON (Serial Control Register) is responsible for all serial communication related settings in 8051.



SCON : Serial Port Control Register (Bit Addressable)

SM0	SM1	SM2	REN	TN8	RB8	TI	RI

SM0	SCON.7	Serial Port mode specifier (NOTE 1).
SM1	SCON.6	Serial Port mode specifier (NOTE 1).
SM2	SCON.5	Enables the multiprocessor communication feature in mode 2 & 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0 (See table 9).
REN	SCON.4	Set/Cleared by software to Enable/Disable reception.
TB8	SCON.3	The 9th bit that will be transmitted in modes 2 & 3. Set/Cleared by software.
RB8	SCON.2	In modes 2 & 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.
TI	SCON.1	Transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes. Must be cleared by software.
RI	SCON.0	Receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or half way through the stop bit time in the other modes (except see SM2). Must be cleared by software.

Note 1 :

SM0	SM1	MODE	DESCRIPTION	BAUD RATE
0	0	0	SHIFT REGISTER	Fosc./12
0	1	1	8 bit UART	Variable
1	0	2	8 bit UART	Fosc./64 OR Fosc./32
1	1	3	8 bit UART	Variable

- First, let's present the "Serial Control" (SCON) SFR and define what each bit of the SFR represents:

Bit	Name	Bit Address	Explanation of Function
7	SM0	9Fh	Serial port mode bit 0
6	SM1	9Eh	Serial port mode bit 1.
5	SM2	9Dh	Mutliprocessor Communications Enable (explained later)
4	REN	9Ch	Receiver Enable. This bit must be set in order to receive characters.
3	TB8	9Bh	Transmit bit 8. The 9th bit to transmit in mode 2 and 3.
2	RB8	9Ah	Receive bit 8. The 9th bit received in mode 2 and 3.
1	TI	99h	Transmit Flag. Set when a byte has been completely transmitted.
0	RI	98h	Receive Flag. Set when a byte has been completely received.

- Additionally, it is necessary to define the function of SM0 and SM1 by an additional table:

SM0	SM1	Serial Mode	Explanation	Baud Rate
0	0	0	8-bit Shift Register	Oscillator / 12
0	1	1	8-bit UART	Set by Timer 1 (*)
1	0	2	9-bit UART	Oscillator / 64 or Oscillator/ 32 (*)
1	1	3	9-bit UART	Set by Timer 1 (*)

- **(*) Note:** The baud rate indicated in this table is doubled if PCON.7 (SMOD) is set.

- The SCON SFR allows us to configure the Serial Port.

SCON bits description
- The first four bits (bits 4 through 7) are configuration bits.

SM0 and SM1 - Serial Mode
- Bits **SM0** and **SM1** let us set the *serial mode* to a value between 0 and 3, inclusive.
- The four modes are defined in the chart immediately above.
- As you can see, selecting the Serial Mode selects the mode of operation (8-bit/9-bit, UART or Shift register) and also determines how the baud rate will be calculated.
- In modes 0 and 2 the baud rate is fixed based on the oscillators frequency.
- In modes 1 and 3 the baud rate is variable based on how often Timer 1 overflows.

SM2 – Multi-processor communication
- The next bit, **SM2**, is a flag for "Multiprocessor communication."
- Generally, whenever a byte has been received the 8051 will set the "RI" (Receive Interrupt) flag.
- This lets the program know that a byte has been received and that it needs to be processed.
- However, when SM2 is set the "RI" flag will only be triggered if the 9th bit received was a "1".
- That is to say, if SM2 is set and a byte is received whose 9th bit is clear, the RI flag will never be set.
- This can be useful in certain advanced serial applications.
- For now it is safe to say that you will almost always want to clear this bit so that the flag is set upon reception of *any* character.

REN

- The next bit, **REN**, is "Receiver Enable."
- This bit is very straightforward: If you want to receive data via the serial port, set this bit.
- You will almost always want to set this bit.
- The last four bits (bits 0 through 3) are operational bits.
- They are used when actually sending and receiving data--they are not used to configure the serial port.

TD8

- The **TB8** bit is used in modes 2 and 3.
- In modes 2 and 3, a total of nine data bits are transmitted.
- The first 8 data bits are the 8 bits of the main value, and the ninth bit is taken from TB8.
- If TB8 is set and a value is written to the serial port, the datas bits will be written to the serial line followed
- by a "set" ninth bit. If TB8 is clear the ninth bit will be "clear."

RD8

- The **RB8** also operates in modes 2 and 3 and functions essentially the same way as TB8, but on the reception side.
- When a byte is received in modes 2 or 3, a total of nine bits are received.
- In this case, the first eight bits received are the data of the serial byte received and the value of the ninth bit received will be placed in RB8.

TI

- **TI** means "Transmit Interrupt."
- When a program writes a value to the serial port, a certain amount of time will pass before the individual bits of the byte are "clocked out" the serial port.
- If the program were to write another byte to the serial port before the first byte was completely output, the data being sent would be garbled.
- Thus, the 8051 lets the program know that it has "clocked out" the last byte by setting the TI bit.
- When the TI bit is set, the program may assume that the serial port is "free" and ready to send the next byte.

RI

- Finally, the **RI** bit means "Receive Interrupt."
- It functions similarly to the "TI" bit, but it indicates that a byte has been received.
- That is to say, whenever the 8051 has received a complete byte it will trigger the RI bit to let the program know that it needs to read the value quickly, before another byte is read.

SCON Register

SM0 / SM1 / SM2 / REN / TB8 / RB8 / TI / RI

SM0 SM1=> mode

0 1 => Mode 1 – start bit + 8b data + stop bit (10b mode)

Setting the Serial Port Baud Rate

- Once the Serial Port Mode has been configured, as explained above, the program must configure the serial ports baud rate.

- This only applies to Serial Port modes 1 and 3.

- The Baud Rate is determined based on the oscillator's frequency when in mode 0 and 2.

    - In mode 0, the baud rate is always the oscillator frequency divided by 12.
    - This means if your crystal is 11.059 MHz, mode 0 baud rate will always be 921,583 baud.

    - In mode 2 the baud rate is always the oscillator frequency divided by 64, so a 11.059Mhz crystal speed will yield a baud rate of 172,797.

- In modes 1 and 3, the baud rate is determined by how frequently timer 1 overflows.
    - The more frequently timer 1 overflows, the higher the baud rate.

- There are many ways one can cause timer 1 to overflow at a rate that determines a baud rate, but the most common method is to put timer 1 in 8-bit auto-reload mode (timer mode 2) and set a reload value (TH1) that causes Timer 1 to overflow at a frequency appropriate to generate a baud rate.

- To determine the value that must be placed in TH1 to generate a given baud rate, we may use the following equation **(assuming PCON.7 is clear)**.
    - ***TH1 = 256 - ((Crystal / 384) / Baud)***

- **If PCON.7 is set** then the baud rate is effectively doubled, thus the equation becomes:

    TH1 = 256 - ((Crystal / 192) / Baud)

- For example, if we have an 11.059 MHz crystal and we want to configure the serial port to 19,200 baud we try plugging it in the first equation:

    TH1 = 256 - ((Crystal / 384) / Baud)
    TH1 = 256 - ((11059000 / 384) / 19200)
    TH1 = 256 - ((28,799) / 19200)
    TH1 = 256 - 1.5 = 254.5

- As you can see, to obtain 19,200 baud on a 11.059Mhz crystal wed have to set TH1 to 254.5.

- If we set it to 254 we will have achieved 14,400 baud and if we set it to 255 we will have achieved 28,800 baud.

    Thus were stuck...

- But not quite... to achieve 19,200 baud we simply need to **set PCON.7 (SMOD)**.

- When we do this we double the baud rate and utilize the second equation mentioned above. Thus we have:

  TH1 = 256 - ((Crystal / 192) / Baud)
  TH1 = 256 - ((11059000 / 192) / 19200)
  TH1 = 256 - ((57699) / 19200)
  TH1 = 256 - 3 = 253

- Here we are able to calculate a nice, even TH1 value.

- Therefore, to obtain 19,200 baud with an 11.059MHz crystal we must:

  1. Configure Serial Port mode 1 or 3.
  2. Configure Timer 1 to timer mode 2 (8-bit auto-reload).
  3. Set TH1 to 253 to reflect the correct frequency for 19,200 baud.
  4. Set PCON.7 (SMOD) to double the baud rate.

Baud Rate

- XTAL frequency = 11.0592
- Machine cycle frequency = 11.0592 / 12 => 921.6 kHz

- 8051 serial communication UART circuitry
- 921.6 kHz / 32 => 28800 Hz => frequency input to Timer 1 to set baud rate

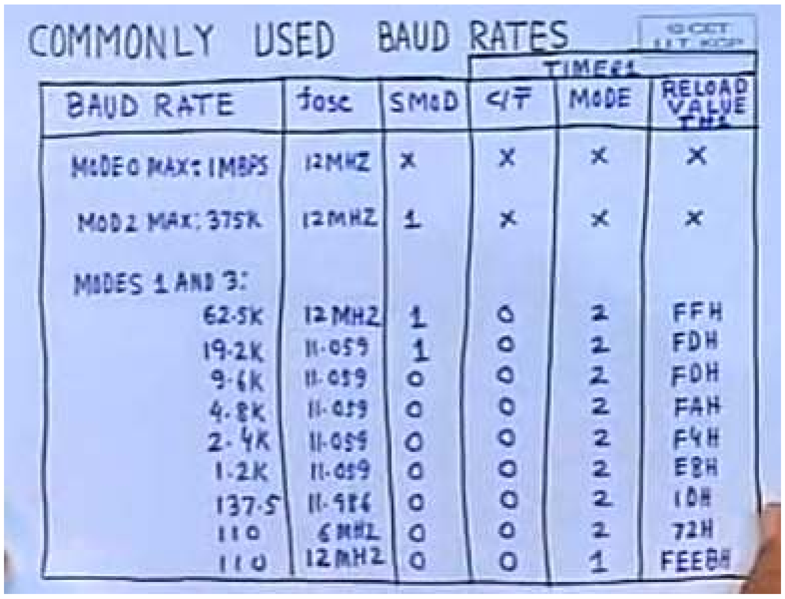
- Timer 1 in mode 2 – 8 b auto reload mode

Finding TH1 value for various baud rates

SMOD = 0 (PCON.7 is clear)
TH1 = 256 – (crystal frequency / 384 * baud_rate)

SMOD=1 (PCON.7 is set – double the baud rate)
TH1 = 256 – (crystal frequency / 192 * baud_rate)

Baud Rate	TH1 (Decimal)	TH1 (Hex)
9600	-3	FD
4800	-6	FA
2400	-12	F4
1200	-24	E8



COMMONLY USED BAUD RATES

BAUD RATE	fosc	SMOD	TIMER1 C/T	TIMER1 MODE	TIMER1 RELOAD VALUE TH1
MODE 0 MAX: 1MBPS	12MHZ	X	X	X	X
MOD 2 MAX: 375K	12MHZ	1	X	X	X
MODES 1 AND 3:					
62.5K	12MHZ	1	0	2	FFH
19.2K	11.059	1	0	2	FDH
9.6K	11.059	0	0	2	FDH
4.8K	11.059	0	0	2	FAH
2.4K	11.059	0	0	2	F4H
1.2K	11.059	0	0	2	E8H
137.5	11.986	0	0	2	1DH
110	6MHZ	0	0	2	72H
110	12MHZ	0	0	1	FEEBH

Calculating Baud Rates

As we know that 8051 microcontrollers takes 12 clock cycles to complete one machine cycle.

So our effective Instruction execution frequency is Fosc/12. If we are using a crystal of 11.0592MHz; our efeective frequency is somewhere around $F_{effective}$= 11.0592/12 MHz => 921.6 KHz.

8051 UART or serial communication block further divide this frequency (921.6 KHz) by 32 to generate its baud rate.

Therefore Effective frequency available to generate Baud rates is 921.6 KHz/32 = 28800 Hz.

So for different standard baud rates the values of TH1 will be

Baud Rate 9600 -- TH1=0xFD --- because 28800/9600 = 3

Serial Buffer Register (SBUF)

SBUF is an 8-bit register used for serial communication specific programs.

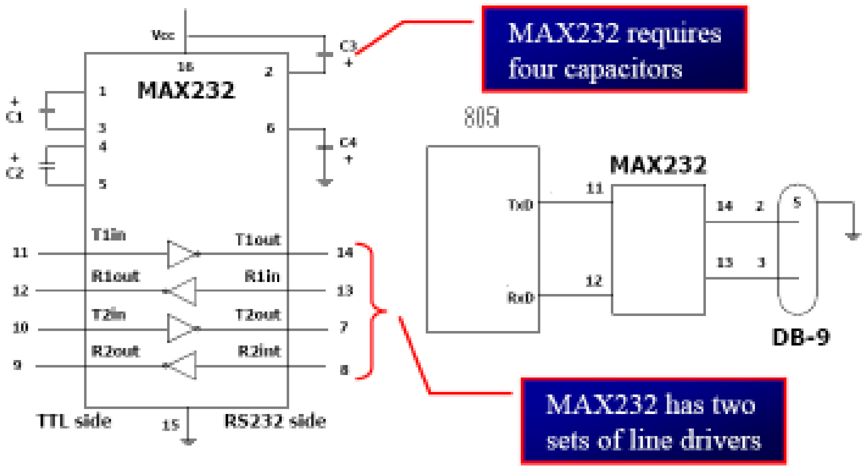
For a byte to to be transferred via TxD line, it must be placed in the SBUF register.

Similarly SBUF holds the byte of data when it is received by 8051's receive line.

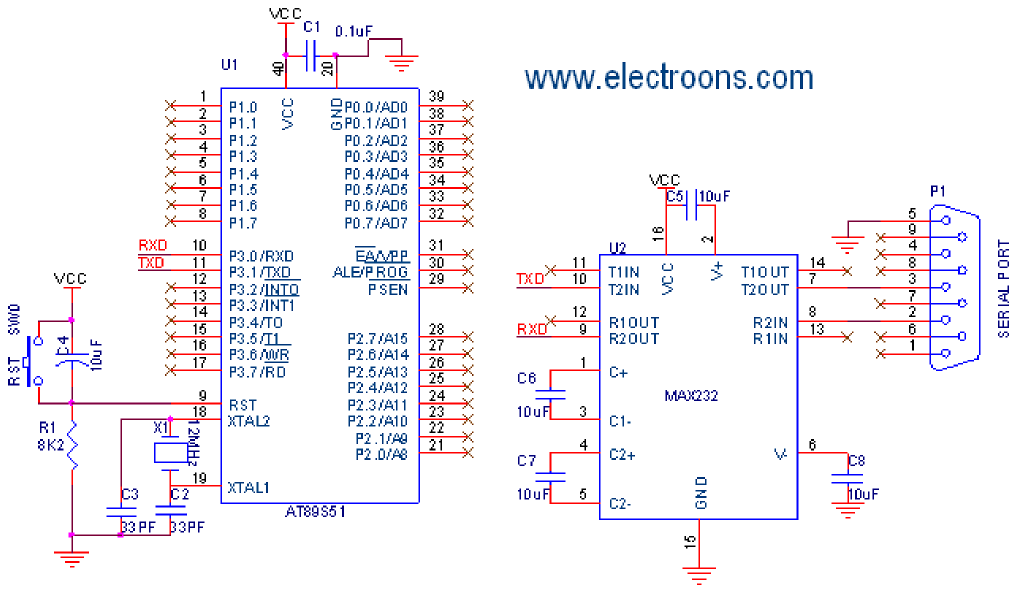
SBUF can be accessed similar to any other register in 8051, but it is not bit addressable.

Hardware Connections

As explained above We need a RS232-TTL level converter to enable 8051 communicate serially with other RS232 compatible devices. Here is the connection schematic...



Final Schematic Diagram -



Writing to the Serial Port

- Once the Serial Port has been properly configured as explained above, the serial port is ready to be used to send data and receive data.

- If you thought that configuring the serial port was simple, using the serial port will be a breeze.

- To write a byte to the serial port one must simply write the value to the **SBUF** (99h) SFR.

- For example, if you wanted to send the letter "A" to the serial port, it could be accomplished as easily as:
    - **MOV SBUF,#A**

- Upon execution of the above instruction the 8051 will begin transmitting the character via the serial port.

- Obviously transmission is not instantaneous--it takes a measurable amount of time to transmit.

- And since the 8051 does not have a serial output buffer we need to be sure that a character is completely transmitted before we try to transmit the next character.

- The 8051 lets us know when it is done transmitting a character by setting the **TI** bit in SCON.

- When this bit is set we know that the last character has been transmitted and that we may send the next character, if any.

- Consider the following code segment:
    - **CLR TI** ;Be sure the bit is initially clear
      MOV SBUF,#A ;Send the letter A to the serial port
      JNB TI,$;Pause until the TI bit is set.

- The above three instructions will successfully transmit a character and wait for the TI bit to be set before continuing.

- The last instruction says "Jump if the TI bit is not set to $"--$, in most assemblers, means "the same address of the current instruction."

- Thus the 8051 will pause on the JNB instruction until the TI bit is set by the 8051 upon successful transmission of the character.

Reading the Serial Port

- Reading data received by the serial port is equally easy.

- To read a byte from the serial port one just needs to read the value stored in the **SBUF** (99h) SFR after the
- 8051 has automatically set the **RI** flag in SCON.

- For example, if your program wants to wait for a character to be received and subsequently read it into the
- Accumulator, the following code segment may be used:
    - **JNB RI,$** ;Wait for the 8051 to set the RI flag
      MOV A,SBUF ;Read the character from the serial port

- The first line of the above code segment waits for the 8051 to set the RI flag; again, the 8051 sets the RI flag automatically when it receives a character via the serial port.

- So as long as the bit is not set the program repeats the "JNB" instruction continuously.

- Once the RI bit is set upon character reception the above condition automatically fails and program flow falls through to the "MOV" instruction which reads the value.

Pseudo code

- Transfer letter 'A' serially at 4800 baud rate continuously using 8b data and 1 stop and start bits

```
o Configure timer 1 in mode 2 (8b auto reload mode)
o Load the timer 1 TH1 8b register with initial value for given baud rate
  (FA for 4800 baud rate)
o Run the timer 1
o Configure the serial port mode to 8b UART with baud rate given by timer 1
  (SCON => SM0 SM1- - - - - - = 0100 0000 => 0x40)
o Do infinitely
    ▪ Send the char 'A' to SBUF
    ▪ Poll on TI flag continuously
    ▪ Reset the TI flag
```

- Transfer message "YES' serially at 9600 baud rate continuously, 8b data, 1 stop bit and start bit

```
o Configure timer 1 in mode 2 (8b auto reload mode)
o Load the timer 1 TH1 8b register with initial value for given baud rate
  (FD for 9600 baud rate)
o Run the timer 1
o Configure the serial port mode to 8b UART with baud rate given by timer 1
  (SCON => SM0 SM1- - - - - - = 0100 0000 => 0x40)
o Do infinitely for chars 'Y' 'E' 'S'
    ▪ Send the char to SBUF
    ▪ Poll on TI flag continuously
    ▪ Reset the TI flag
```

- Echo back the user input using serial port (use polling)

```
o Configure timer 1 in mode 2 (8b auto reload mode)
o Load the timer 1 TH1 8b register with initial value for given baud rate
  (FA for 4800 baud rate)
o Run the timer 1
o Configure the serial port mode to 8b UART with baud rate given by timer 1
  Set the receive enable bit
  (SCON => SM0 SM1 - REN - - - - = 0101 0000 => 0x50)
o Do infinitely
    ▪ Poll on RI flag continuously
    ▪ receive the byte from SBUF  //receive the user input char from serial port
    ▪ reset the RI flag
    ▪ Send the received byte to SBUF //echo back the received input char back to
      serial
    ▪ Poll on TI flag continuously
    ▪ Reset the TI flag
```

- Using printf() from standard io libraries – stdio.h
    - Include target header file - special function register declarations for the intended 8051 derivative
    - Include standard io header filel for prototype declarations of io functions
    - *Configure timer 1 in mode 2 (8b auto reload mode)*
    - *Load the timer 1 TH1 8b register with initial value for given baud rate*
      (FA for 4800 baud rate)
    - *Run the timer 1*
    - *Configure the serial port mode to 8b UART with baud rate given by timer 1*
      Set the receive enable bit
      (SCON => SM0 SM1 - REN - - - - = 0101 0000 => 0x50)
    - *Do infinitely*
        - *Use Printf() to display any message on serial port*

- Echo back the user input using serial port (use ISR)
    - *Configure timer 1 in mode 2 (8b auto reload mode)*
    - *Load the timer 1 TH1 8b register with initial value for given baud rate*
      (FA for 4800 baud rate)
    - *Run the timer 1*
    - *Configure the serial port mode to 8b UART with baud rate given by timer 1*
      Set the receive enable bit
      (SCON => SM0 SM1 - REN - - - - = 0101 0000 => 0x50)
    - *Enable global interrupt*
    - *Enable Serial Interrupt (Interrupt source of interest)*
    - *Write an ISR for serial interrupt (irq# 4)*
        - *Check for transmit interrupt flag (TI)*
            - *Reset the TI interrupt flag if it is set*
        - *Check for receive interrupt flag (RI)*
            - *Read the byte from SBUF*
            - *Reset the RI interrupt flag*
            - *Echo back the received byte to serial port (SBUF)*

- Transfer letter 'A' serially at 4800 baud rate continuously using 8b data and 1 stop and start bits

```
//Transfer letter 'A' serially at 4800 baud rate continuously.
//USe 8b data and 1 stop bit
#include <at89c51xd2.h>
void main(void)
{
	TMOD = 0x20;   //Timer 1 in mode 2 8b auto reload
	TH1 = 0xFA;
	TR1 = 1;

	SCON = 0x40;

	while(1)
	{
		SBUF = 'A';
		while(TI == 0);
		TI=0;
	}
}
```

- Transfer message “YES' serially at 9600 baud rate continuously, 8b data, 1 stop bit and start bit

```
#include <at89c51xd2.h>

void SerTx(unsigned char);

void main(void)
{
        TMOD = 0x20; // Timer 1, 8b auto reload mode
        TH1 = 0xFD;      //9600 baud rate
        TR1 = 1;

        SCON = 0x40;

        while(1)
        {
                SerTx('Y');
                SerTx('E');
                SerTx('S');
        }
}

void SerTx(unsigned char byte)
{
        SBUF = byte;
        while(TI == 0);
        TI=0;
}
```

Echo back the user input using serial port (use polling)

```
#include <at89c51xd2.h>
void main(void)
{
        unsigned char rx_byte;

        TMOD = 0x20; //timer 1 in mode 2 auto reload mode
        TH1 = 0xFA;
        TR1 = 1;

        SCON = 0x50;

        while(1)
        {
                while(RI == 0);
                rx_byte = SBUF;
                P1 = rx_byte;
                RI=0;

                SBUF = rx_byte;
                while(TI == 0);
                TI = 0;
        }
}
```

Using printf() from standard io libraries – stdio.h

```
#include <REG52.H>              /* special function register declarations   */
                                /* for the intended 8051 derivative         */

#include <stdio.h>              /* prototype declarations for I/O functions */

/*------------------------------------------------
The main C function.  Program execution starts
here after stack initialization.
------------------------------------------------*/
void main (void)
{

      SCON  = 0x50;                       /* SCON: mode 1, 8-bit UART, enable rcvr
    */
      TMOD |= 0x20;               /* TMOD: timer 1, mode 2, 8-bit reload        */
      TH1   = 221;                /* TH1:  reload value for 1200 baud @ 16MHz   */
      TR1   = 1;                  /* TR1:  timer 1 run                          */

    /*------------------------------------------------
    Note that an embedded program never exits (because
    there is no operating system to return to).  It
    must loop and execute forever.
    ------------------------------------------------*/
     while (1)
    {
      TI=1;
      P1 ^= 0x01;                 /* Toggle P1.0 each time we print */
      printf ("Hello World\n");   /* Print "Hello World" */
     }
}
```

Echo back the user input using serial port (use ISR)

```
//Transfer letter 'A' serially at 4800 baud rate continuously.
//USe 8b data and 1 start and stop bit
#include <at89c51xd2.h>
void main(void)
{
        /* UART Initialization for 9600 Baud Rate */
        SCON = 0x50;                // 8 bit mode, receiver enabled
        TMOD |= 0x20;               // Timer-1 selected as Auto reload mode
        TH1  = 0xfd;                // 0xfd for 11MHz, oxb2 for 24MHz
        TR1  = 1;                   // Start Timer-1

        //enable the interrupts - global access to interrupts and interrupt from serial
        //EA - - ES ET1 EX1 ET0 EX0
        //1  0 0  1  0   0    0  0
        //0x90
        //IEN0 = 0x90     or do it using individual bit registers as shown below

        EA = 1;                 // Enable global interrupt
        ES = 1;                 // Enable Serial Port Interrupt
}

/* UART Interrupt Service Routine */
void uart_isr(void) interrupt 4
{
        unsigned char rx_byte;
        if(TI == 1)                 // Check for Transmit interrupt
        {
                TI = 0;             // Clear Transmit interrupt flag
        }
        else if(RI == 1)            // Check for Receive interrupt
        {
                rx_byte = SBUF;
                P1 = rx_byte;
                RI=0;
                SBUF = rx_byte;
        }
}
```

```
/*********************************************************************
>Example program to send some characters serially at 9600 bps.
>Make all connection according to the schematic given above.
>Serial port Mode 1 is used with 8bit data, 1 stop bit, 1 start bit
>One important thing is that all calculations for baud rate generation using Timer1 are made for Timer1 8
bit auto reload mode
*********************************************************************/


#include<at89x52.h>

void main(void)
{
   TMOD=0x20; // Timer1 Mode2 8 bit auto reload
   TH1=0xFD; // 9600 bps

   SCON=0x50; // 8 Data bit, 1 start bit, 1 stop bit
   TR1=1; // Timer1 ON

    while(1==1)
   {
     SBUF='S';
     while(TI==0); // Pole TI flag for complete transmission
     TI=0;
     SBUF='A';
     while(TI==0);
     TI=0;
     SBUF='M';
     while(TI==0);
     TI=0;
    }
}
```

```
/******************************************************************
>Example program to send a string serially at 9600 bps.
>Make all connection according to the schematic given above.
>Serial port Mode 1 is used with 8bit data, 1 stop bit, 1 start bit
>One important thing is that all calculations for baud rate generation using Timer1 are made for Timer1 8
bit auto reload mode
******************************************************************/


#include<at89x52.h>

void serial(unsigned char x)
{
   SBUF=x;
   while(TI==0);
   TI=0;
}


void rs_puts(char *aaa)
{
    unsigned int i;
    for(i=0;aaa[i]!=0;i++)
   {
       serial(aaa[i]);
   }
}

void main(void)
{
   TMOD=0x20; // Timer1 Mode2 8 bit auto reload
   TH1=0xFD; // 9600 bps

   SCON=0x50; // 8 Data bit, 1 start bit, 1 stop bit

   TR1=1; // Timer1 ON

    while(1==1)
   {
     rs_puts("www.electrroons.com\n\r");
    }
}
```