A Simple RS232 Guide

Posted on 12/12/2005
By
Jon Ledbetter
Updated 05/22/2007

The reason behind it

I assembled this guide from various posts and links from 8052.com It is meant to serve as a simple guide to getting a working RS232 communications connection between a personal computer (PC) and a micro controller based device. Despite the wealth of information available on the web and 8052.com, there are still many, many very basic questions posted every week. This was designed to give everything all in one place, so that there doesn't have to be a here and there search for all the relevant information. Several members on the 8052.com forum contributed to the information contained herein.

Thanksgiving

Jan Waclawek, Slobodan Madaric, Andy Neil, Sasha Jevtic, Steve Taylor, Erik Malund, among others for the comments, suggestions, previous posts and knowledge.

And, of course special thanks to Craig Steiner for giving us this great website and forum as well as the inclusion of his "Serial Port Operation" tutorial.

Onward

The first section of this guide will cover terminal programs and specifically the setup/configuration of Hyper-Terminal. I use this terminal program without any problems, however a lot of people do not like it. It is on most every Windows PC that exists. Aside from that, you can use any terminal program you like, but you'll have to figure out where the settings are and how to change them on your own.

The second section will cover level converters and shows the basic wiring and testing of the MAX232 and 8052 micro controller. There are a few different diagrams, all of which show basically the same information. There are some hardware tests and software tests at the end of the section.

The last section contains further reading, links to the reference threads from which this information was taken, links to tutorials and other information that will give more in depth understanding of RS232 and what it is, as well as Craig Steiner's Chapter 8 "Serial Port Operation" (with his permission)

Terminal Programs

Terminal programs or terminal emulators come in many different flavors. I will mention some links to a few different programs, but if you choose to use one of these or something else, you will have to read the documentation that comes with it to learn where and how to change the settings. I will only be covering Hyper-Terminal because most people already have it if they have a "Windows" computer.

The terminal program's basic function is to receive and transmit data to and from the micro controller. There are more specific terms as to the connection/configuration type, which you can learn more about here.

We will specifically cover a DCE at 9600 baud, eight bit data, no parity and one stop bit. So in your terminal program the objective will be to set the following parameters.:

Baud Rate: 9600
Data Bits: 8
Parity: none or 0
Stop Bits: 1
Flow Control: None

Bray http://bray.velenje.cx/avr/terminal/

Tera Terminal http://hp.vector.co.jp/authors/VA002416/teraterm.html

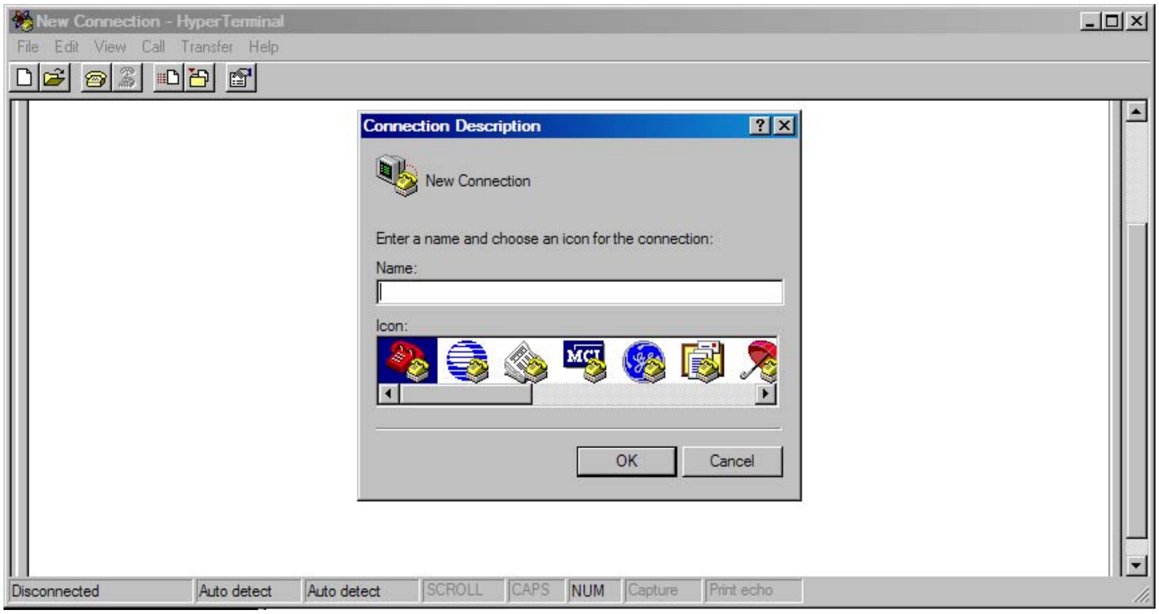
Procomm http://www.symantec.com/procomm/

For Linux http://sourceforge.net/projects/ltsp/

For Macintosh
http://www.retards.org/library/technology/computers/terminals/macintosh.html

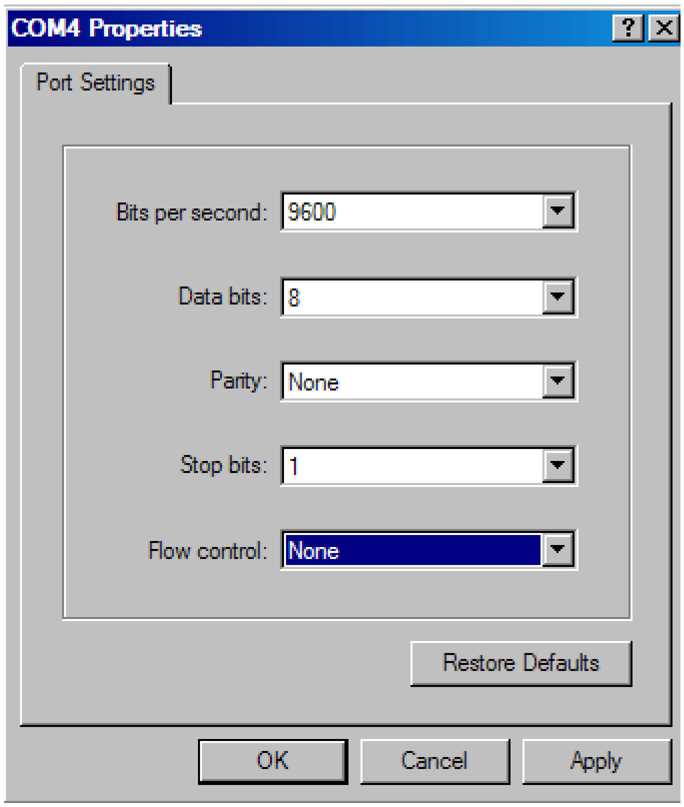
Configuring Hyper Terminal

First, start Hyper-terminal. Go to File> New Connection and you will see the following dialog box.

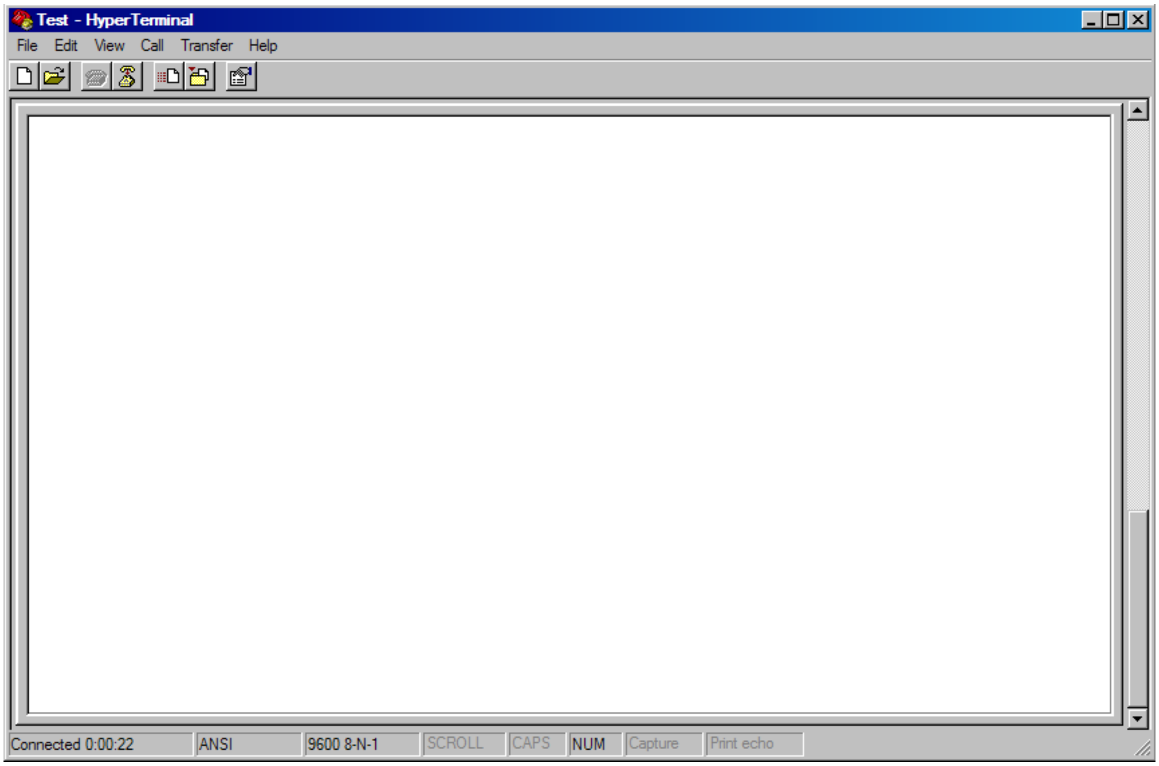


Enter a name for your new connection and click "OK". In the next dialog box, go to the "Connect Using" field and select your available Comm. Port, then click "OK".

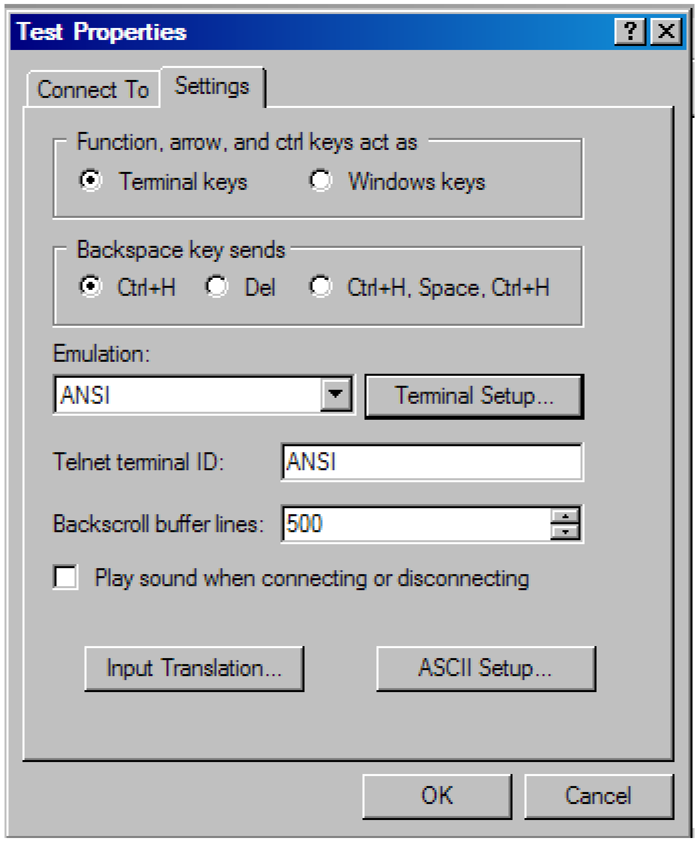
The next dialog box is where we setup our port settings. Using the drop down list boxes, make your settings match the figure below. Click "OK" to proceed.



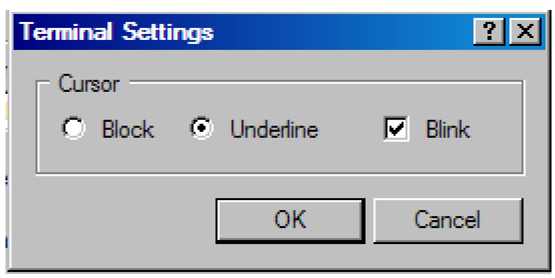
You should now be back to a blank terminal screen, like below. Now go to File> Properties.



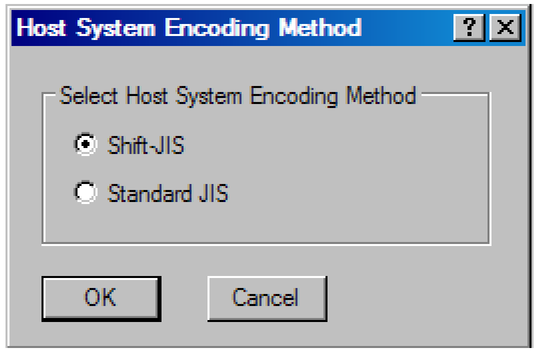
Click on the “Settings” tab, and using the drop down list box, under Emulation, select “ANSI” as shown below. Then click the “Terminal Setup” button to the right.



Nothing Critical here, make your setting match the ones below and click “OK”

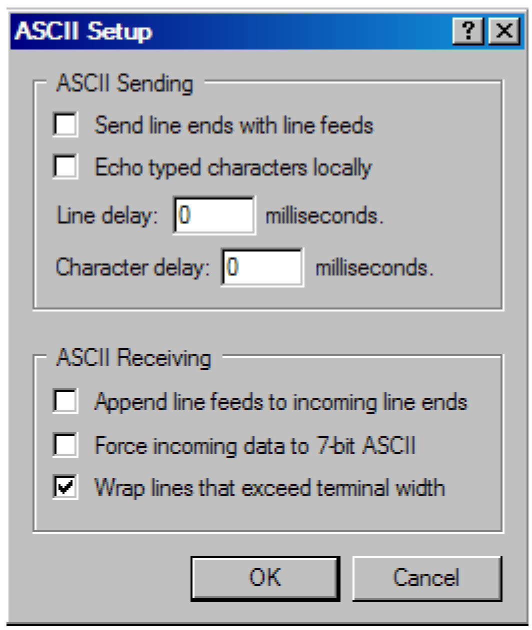


Now click on the “Input Translation” button.



Again, match your settings to the ones above and click “OK”

Now click on the “ASCII Setup” button and you will see the following dialog box.



Besides the Port Settings we set earlier, this dialog box will make the biggest difference in how your session will work.

If your micro application is looking for a CR or carriage return to signal the end of a line input, then you will probably need to check the “Send line ends with line feeds” box.

If your micro application does not “echo” characters back to the terminal and you do not see anything when you type, then you will need to check the “Echo typed characters locally” box.

When your micro application sends data back to the terminal, if all the lines are end to end like this:

Hello WorldHello WorldHello World

And the desired display should look like this:

Hello World
Hello World
Hello World

Then you will need to check the "Append line feeds to incoming line ends" box. On the other hand, if this box is checked and your display looks like this:

Hello World

Hello World

Hello World

Then you will want to uncheck the "Append line feeds to incoming line ends" box.

Note: *Some versions of Hyper-Terminal apparently don't actually update the settings until, you disconnect and then re-connect. So you should follow this procedure to make sure your changed settings are used.*

To see how this works, let's construct a simple loop back tester. Using a female DB-9 or DB-25 connector, jumper pins 2 and 3 together. Put this connector on the end of your serial cable coming from your PC. This will echo all characters typed in Hyper-Terminal back to Hyper-Terminal. Play with the above settings to see how they work.

(Note: Something worth remembering. In the above dialog box there is a line and character delay. For most purposes these should be set to zero, there is a time when these can come in handy. Say that your micro controller can't process characters as fast as they are being sent, and the micro occasionally misses a character. To give the micro more time to process each character, you can put a delay in between each character.)

Level Converters

A few comments on level converters, level translators or line drivers as they are called. The micro controller UART TXD/RXD pins are TTL/CMOS voltage levels. The specification for RS232 is basically -3 to -25 volts for "1" and +3 to +25 volts for a "0". Therefore we need some way to convert from one to the other and vice versa, this is where the MAX232 comes in. It is a RS232 transceiver that will convert the voltage levels for us.

There are several brands and versions, we will deal with three variations of the Maxim brand. In the first schematic below, the part is MAX232A which uses five .1uF caps, in the second part is a MAX232 which uses five 1uF caps and the third is MAX233A which only uses one 1uF cap. One draw back to the MAX233A is that it costs about twice what the MAX232 and MAX232A cost. Which ever brand and variation you decide to use, if different from the three I've already mentioned, be sure and read the datasheet to find out the correct wiring and component values. For now I suggest you use the MAX232A until you get this first project working.

Note: *There are several circuits floating around the internet that claim to be tried and true methods of getting by without a MAX232 or equivalent converter, most of these are total junk and you are encouraged to not to waste your time with such circuits.*

Additional Information on RS232 and standards

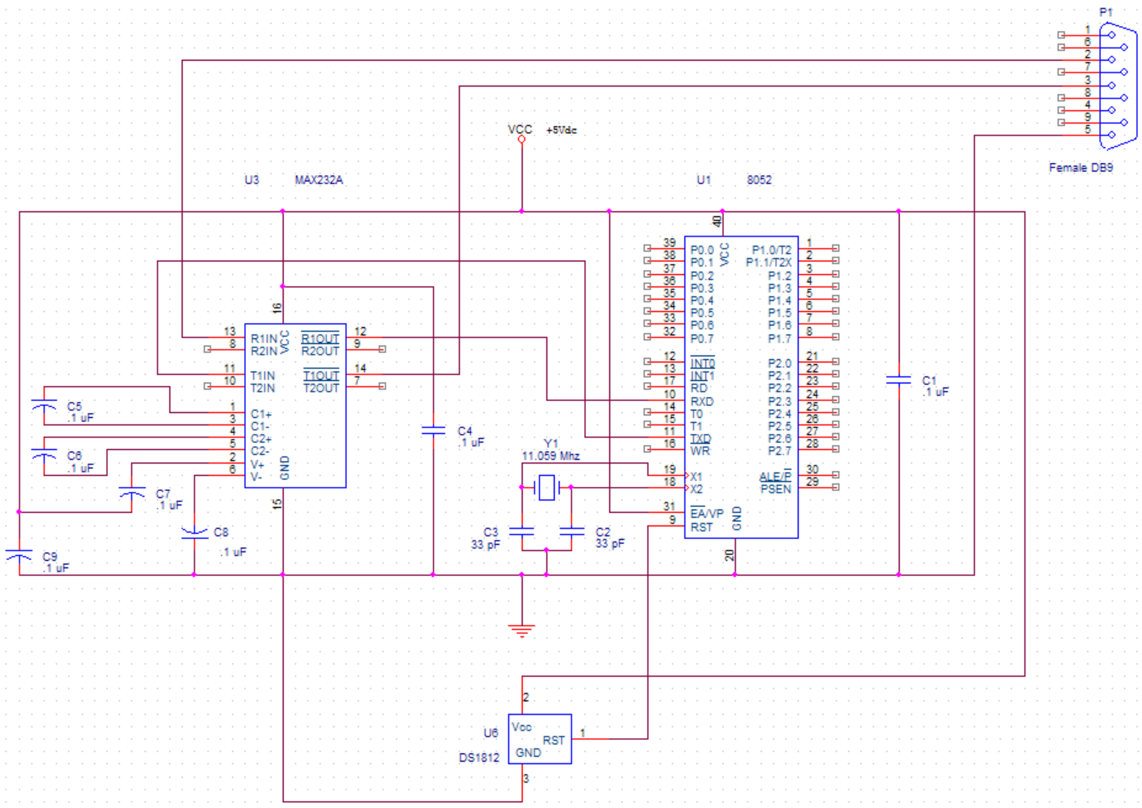
Craig Peacock's Serial Guide http://www.beyondlogic.org/serial/serial.pdf

Telecommunications Industry Association http://www.tiaonline.org

http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html

http://www.taltech.com/TALtech_web/resources/intro-sc.html

Now if everything is working so far, we need test the serial hardware on the controller project. Now wire up the MAX232A to your 8051/8052 like shown below. This is the bare minimum of what you will need to accomplish simple RS232 communication. This is a DCE wiring. For a DTE configuration you would use a Male DB connector and the wires on pins 2 and 3 would be reversed. The DTE also requires a different cable.



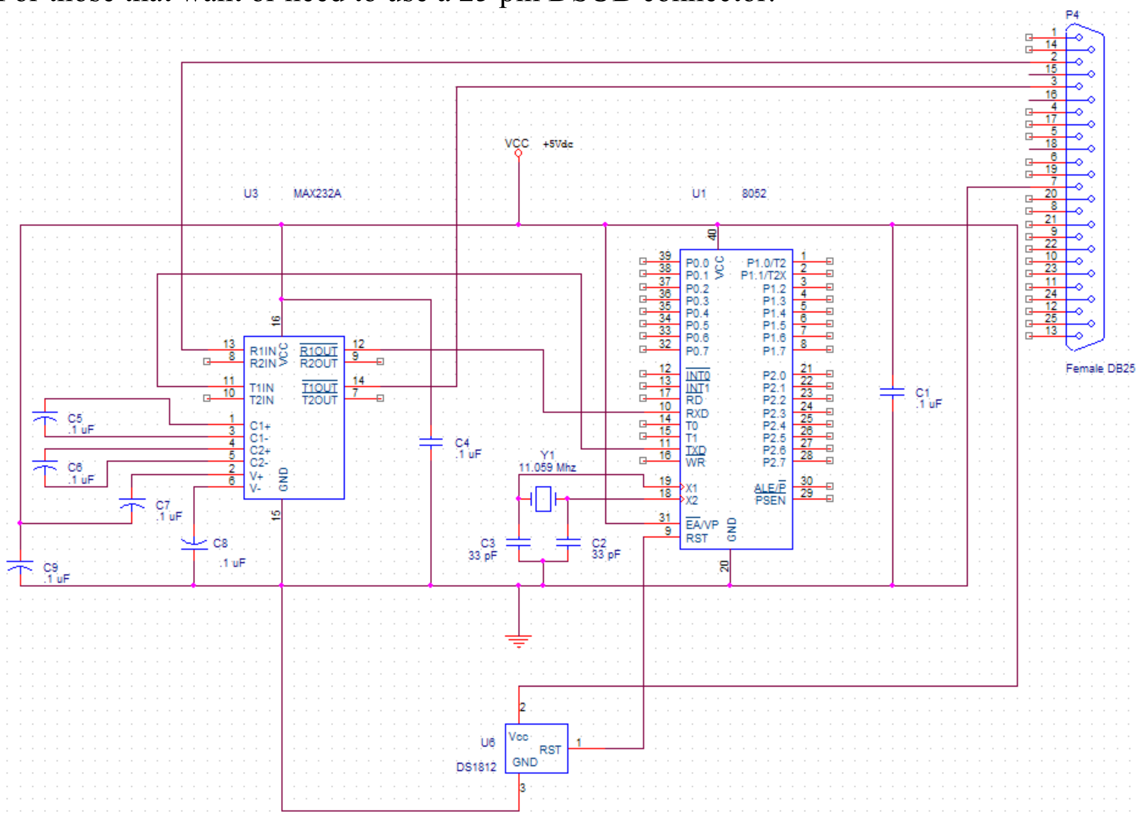
Then your cable should be wired like below. For initial testing, only make it 3 feet long. This is a DCE cable. For DTE you would use a Female connector on both ends and a cross-wiring of pins 2 and 3, also know as a "Null Modem" cable.

2 to 3

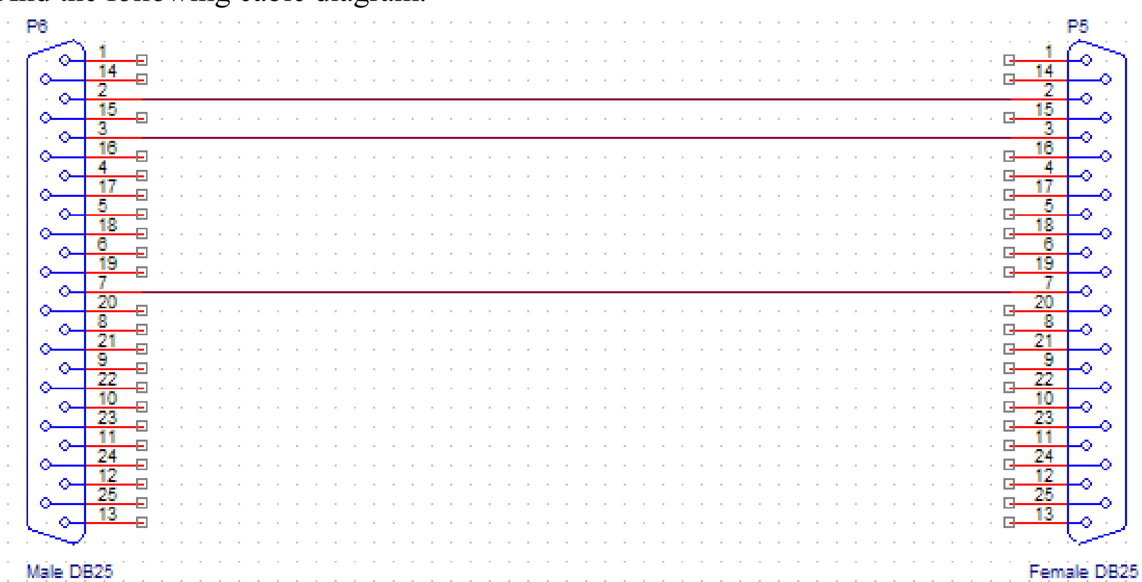
3 to 2

Ground to Ground

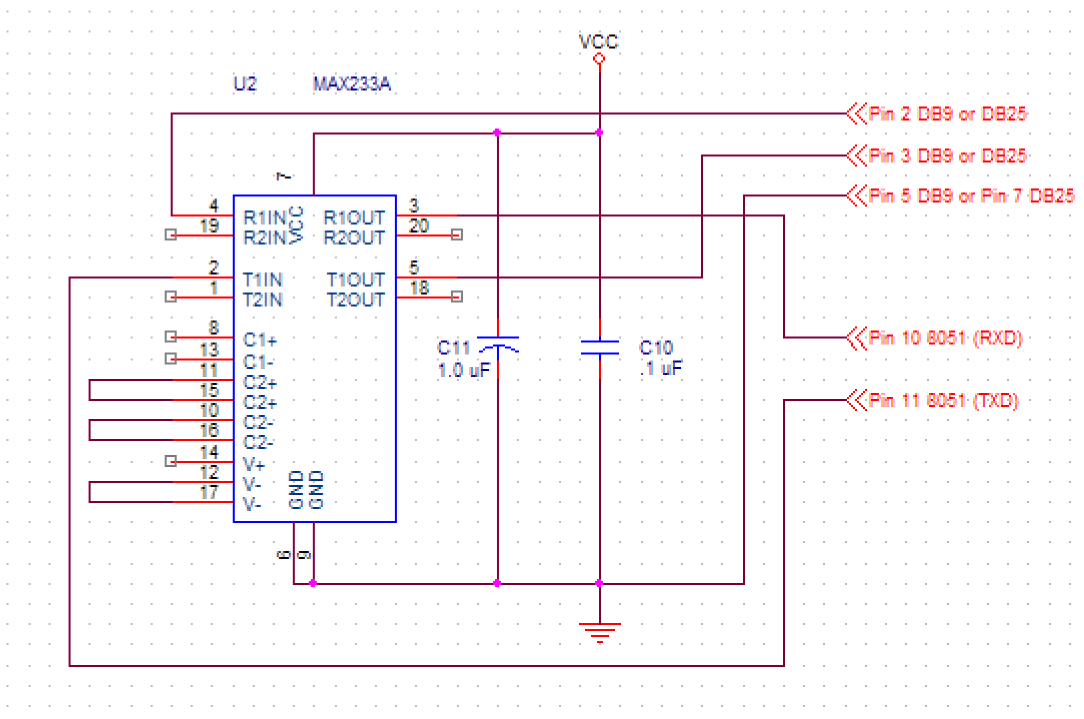
For those that want or need to use a 25 pin DSUB connector.



And the following cable diagram.



Now there are those of you that may have a cap-less version of the MAX232 which would be MAX233A that can refer to the following diagram. I don't know how popular the cap-less versions are at almost twice the cost, but simplicity of wiring and the PCB space saved will be the deciding factors.



Program you micro controller with the first of the following small programs. The first routine will work for any baud rate. This routine will loop data from the terminal program through the micro controller back to the terminal program, just like when you used the loop back connector earlier. This routine will verify that your hardware is wired properly and working.

Test Program 1

```
        ORG     0
Loop:
        MOV     C,P3.0
        MOV     P3.1,C
        SJMP    Loop
```

If every thing is good so far, program your controller with the next short program. This will do all the same things as the above program plus verify that the UART itself is working and properly configured at 9600 baud.

Test Program 2

```
        ORG     0
        LJMP    BEGIN
        ORG     40h
BEGIN:  MOV     SCON,#01110000B ; Mode 1/Stop b/Rec en/x/x/Flags
        MOV     TH1,#0FDH       ; Reload value for 9600 Bd
        MOV     TCON,#01010101B ; Fl1/Tim1 on/Fl0/Tim0 on/Ext1-
                                ; edge/Ext0-edge
        MOV     TMOD,#00100001B ; Gate1/Timer/Mode 2 / Gate0/Timer/Mode
                                ; 1
START:  JNB     RI,START
        CLR     RI
        MOV     A,SBUF
        MOV     SBUF,A
        AJMP    START
```

Alternately you can replace the 'START' section of the above program with the section below. In this configuration the repeating loop will send a continuous string of 'U's (55H) to the terminal program, which when viewed on an oscilloscope will show an alternating pattern of 1's and 0's resulting in a 50% duty cycle square wave at a frequency of 9600/2 = 4800 Hz, for analysis purposes.

Test Program 3

```
START:  MOV     SBUF,#55H
SERWT:  JNB     SCON.1, SERWT
        CLR     SCON.1
        AJMP    START
```

If you've made it this far, you now have a working RS232 connection between your micro controller and your PC terminal program. In the next example I'll show you how to send simple messages to the terminal program from your micro controller project.

Now program your micro controller with the following example.
(Note: The following examples were assembled and simulated with Pinnacle52)

Example Program 1 - polled transmit

```
;*****************************************************************
;This is an example of polled transmit
;*****************************************************************
;
CR      EQU     0DH
LF      EQU     0AH
;
        ORG     0
        LJMP    MAIN
        ;
        ORG     40H
MAIN:   LCALL   SERINIT
        MOV     DPTR,#HELLO
        LCALL   TEXT_OUT
        MOV     DPTR,#MSG1
        LCALL   TEXT_OUT
        MOV     DPTR,#MSG2
        LCALL   TEXT_OUT
LOOP:   AJMP    LOOP                    ;STOP HERE
;*****************************************************************
;Other Serial Messages
;
HELLO:  DB      'Hello World',CR,LF,0
MSG1:   DB      'I am sending messages to my terminal!',CR,LF,0
MSG2:   DB      'This is fun..',CR,LF,0
;*****************************************************************
SERINIT:        ;Initialize serial port
        CLR     TR1
        CLR     TI
        CLR     RI
        MOV     SCON,#01011010B ;TI SET INDICATES TRANSMITTER READY.
                                ;MODE 1 REN
        MOV     TMOD,#00100001B ;TIMER 1 MODE 8 BIT AUTO RELOAD
        MOV     TH1,#0FDH       ;TIMER RELOAD VALUE
        SETB    TR1             ;START TIMER
        MOV     DPTR,#SINIT
        LCALL   TEXT_OUT
        RET
;
SINIT:  DB      CR,LF
        DB      'Serial Port Initialized! '
CRLF:   DB      CR,LF,0
;*****************************************************************
TEXT_OUT:
WT1:    CLR     A
        MOVC    A,@A+DPTR
        INC     DPTR
        JZ      WT2
        LCALL   CHAR_OUT
        AJMP    WT1
WT2:    RET
;*****************************************************************
CHAR_OUT:
```

```
        CLR     TI
        MOV     SBUF,A
        JNB     TI,$
CORET:  RET
;*****************************************************************
```

The output should look like this:

Serial Port Initialized!
Hello World
I am sending messages to my terminal!
This is fun.

Example Program 2 - polled transmit and polled receive

```
;*****************************************************************
;This is an example of polled transmit and receive
;*****************************************************************
;
CR     EQU   0DH
LF     EQU   0AH
;
       ORG   0
       LJMP  MAIN
       ;
       ORG   40H
MAIN: LCALL SERINIT
LOOP: LCALL GETCMD
       ;DO SOME OTHER
       ;STUFF HERE
       AJMP  LOOP
;*****************************************************************
;THIS ROUTINE PROCESSES THE COMMANDS RECEIVED FROM PC
;
GETCMD:
       MOV   A,#0
       ACALL CHAR_IN     ;WAIT FOR PC TO SENT A CHARACTER
                         ;NOW PROCESS
       CJNE  A,#'1',NOT1 ;IF ITS NOT 1 THEN CHECK NEXT POSSIBILITY
       ACALL SUB1        ;ELSE ITS 1
NOT1: CJNE  A,#'2',NOT2 ;IF ITS NOT 2 THEN CHECK NEXT POSSIBILITY
       ACALL SUB2        ;ELSE ITS 2
NOT2: CJNE  A,#'3',NOT3 ;IF ITS NOT 3 THEN CHECK NEXT POSSIBILITY
       ACALL SUB3        ;ELSE ITS 3
NOT3:
ENDCHK:
       RET
;*****************************************************************
SUB1: MOV   DPTR,#MSG1
       ACALL TEXT_OUT
       RET
SUB2: MOV   DPTR,#MSG2
       ACALL TEXT_OUT
       RET
SUB3: MOV   DPTR,#MSG3
```

```
      ACALL TEXT_OUT
      RET
;*****************************************************************
MSG1: DB    'You pressed one',CR,LF,0
MSG2: DB    'You pressed two',CR,LF,0
MSG3: DB    'You pressed three',CR,LF,0
;*****************************************************************
SERINIT:    ;Initialize serial port
      CLR   TR1
      CLR   TI
      CLR   RI
      MOV   SCON,#01011010B   ;TI SET INDICATES TRANSMITTER READY.
                              ;MODE 1 REN
      MOV   TMOD,#00100001B   ;TIMER 1 MODE 8 BIT AUTO RELOAD
      MOV   TH1,#0FDH         ;TIMER RELOAD VALUE
      SETB  TR1               ;START TIMER
      MOV   DPTR,#SINIT
      LCALL TEXT_OUT
      RET
;
SINIT:      DB    CR,LF
      DB    'Serial Port Initialized! '
CRLF: DB    CR,LF,0
;*****************************************************************
TEXT_OUT:
WT1:  CLR   A
      MOVC  A,@A+DPTR
      INC   DPTR
      JZ    WT2
      LCALL CHAR_OUT
      AJMP  WT1
WT2:  RET
;*****************************************************************
CHAR_IN:
      JNB   RI,CHAR_IN
      MOV   A,SBUF
      CLR   RI
      RET
;*****************************************************************
CHAR_OUT:
      CLR   TI
      MOV   SBUF,A
      JNB   TI,$
CORET:      RET
;*****************************************************************
```

Interrupt Driven Serial, Ring Buffers and Flow Control

While polled transmit and receive can handle a great deal of applications, there are times when you may need for your micro controller to be doing something else while receiving or transmitting data. This is where interrupt driven serial communications comes in. Interrupt driven serial routines involve much more overhead as far as program code size and RAM requirements, but have more flexibility to allow the micro controller to do other things while sending and/or receiving data, plus insurance that received information is less likely to be missed.

In the previous examples where polled transmit and receive were used, the programmer must know in advance when information is going to be received. With the interrupt driven communications, all we need to do is check the status of the receive buffer occasionally and process the information if any. As far as transmitting, just put the data in the transmit buffer and continue doing whatever else needs to be done.

Buffers are areas of ram that are used to hold data when sending and receiving. Buffers can be one byte to hundreds of bytes long. This will mainly depend on the amount of RAM you can allocate for the purpose. Ring buffers, circular buffers, and FIFO buffers all mean that the first data put into the buffer will be the first data taken out of the buffer. When the buffer reaches the end then it circles back to the beginning. If these buffers wrap around before the data can be taken out, the previous data is over written and lost. In most cases both buffers should be at least as big as the largest data string to be sent or received. If the buffers are smaller then some sort of buffer flow control may be needed. Flow control is a method of stopping data flow when the buffers are nearly full and continuing data flow when the buffers are nearly empty. This is also known as hand-shaking, and can be done via hardware with the RTS/CTS and DSR/DTR or through software with XON/XOFF. Flow control is only mentioned here to tell you what it is, and will not have any examples at this point.

Example Program 3 - Interrupt Driven Transmit and Receive (with ring buffers)

```
;*******************************************************************
;*******************************************************************
CR          EQU   0DH
LF          EQU   0AH
            ;
TXBUFSZ     EQU   2           ;Define buffer sizes
RXBUFSZ     EQU   8
            ;                 ;Indexes
RX_IN       EQU   08H         ;RX next to go in
RX_OUT      EQU   09H         ;RX next to go out
TX_IN       EQU   0AH         ;TX next to go in
TX_OUT      EQU   0BH         ;TX next to go out
RXBUF       EQU   0CH         ;Allow RXBUFSZ to next variable
TXBUF       EQU   14H         ;Allow TXBUFSZ to next variable
            ;
CHR         EQU   16H
            ;
NEEDTI      BIT   01H         ;Clear it if TI=1 is needed
```

```
            ;                 ;Set if TI=1 is not needed
;*******************************************************************
      ORG   0h
      LJMP  MAIN              ;Reset vector
      ;
;*****************************************************************
      ORG   23h
      LJMP  SER_ISR           ;Serial port interrupt vector
      ;
;*******************************************************************
      ORG   40h
      ;
MAIN: MOV   SP,#17h           ;Initialize stack pointer
      LCALL SERINIT           ;Initialize serial port
      SETB  EA                ;Enable all interrupts
      MOV   DPTR,#SINIT       ;Serial Port Initialized!
      LCALL TEXT_OUT          ;Send the message out
;
LOOP:                         ;Repeating loop
      LCALL GETCHR
      MOV   A,R1
      CPL   A
      JZ    LOOP
      LCALL PUTCHR
      AJMP  LOOP              ;Loop back
;
;*******************************************************************
TEXT_OUT:
      CLR   A                 ;dptr has message
      MOVC  A,@A+DPTR         ;get the next character
      INC   DPTR              ;move index to next position
      JZ    WT2               ;if zero then end of message
      MOV   R1,A              ;else put the chr in the tx buffer
      LCALL PUTCHR
;################################
      mov   r3,#2             ;Temporary delay loop
LP1:  mov   r2,#225           ;to prevent buffer overrun
      djnz  r2,$
      djnz  r3,LP1
;################################
      AJMP  TEXT_OUT
WT2:  RET
;
;*******************************************************************
;                             ;R1 has character
PUTCHR:
      SETB  C                 ;using C to store state of EA
      JBC   EA,PCSKP          ;if interrupts enabled, then disable
      CLR   C                 ;EA was already disabled
      ;
PCSKP:      PUSH  PSW         ;store current status, including EA
      MOV   CHR,R1            ;store the character
      CLR   C
      MOV   A,TX_IN
      SUBB  A,TX_OUT
      CLR   C
      SUBB  A,#TXBUFSZ        ;TXBUFSZ = 2
      JC    TBUFOK            ;
      MOV   R1,#0FFh          ;else error RETURN VALUE -1
      AJMP  PCXIT             ;buffer full  return
```

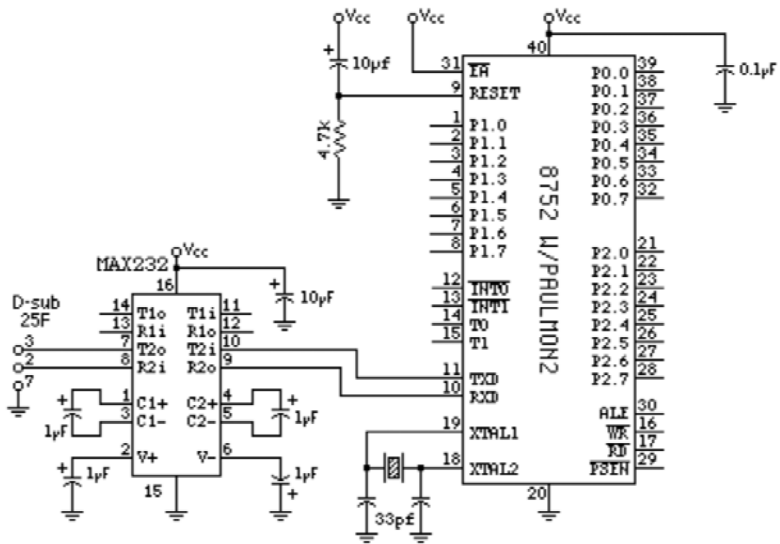
```
      ;
TBUFOK:
      MOV   A,TX_IN
      ANL   A,#TXBUFSZ-1       ;TXBUFSZ =2-1=1 0 through 1 = 2
      ADD   A,#TXBUF           ;# of characters in buffer+buffer address
      MOV   R0,A               ;location in buffer
      MOV   @R0,CHR            ;put character in buffer
      INC   TX_IN              ;move index
      JNB   NEEDTI,SKPTI       ;exit if no more characters in buffer
      CLR   NEEDTI             ;else more to send
      SETB  TI                 ;interrupt and send them
SKPTI:
      MOV   R1,#00h            ;done - return value 0
PCXIT:
      POP   PSW                ;restore and return
      MOV   EA,C
      RET
;
;*********************************************************************
GETCHR:
      SETB  C                  ;use C to store the state of EA
      JBC   EA,GCSKP           ;if all enabled then disable
      CLR   C                  ;all were disabled already
GCSKP:
      PUSH  PSW                ;store the current state
      ;
      CLR   C
      MOV   A,RX_IN            ;how many characters in buffer?
      SUBB  A,RX_OUT
      JNZ   RBUFNZ             ;jump if buffer not empty
      MOV   R1,#0FFh           ;else buffer is empty return value -1
      AJMP  GCXIT              ;exit with buffer empty code
      ;
RBUFNZ:
      MOV   R1,RX_OUT          ;next to get out
      INC   RX_OUT             ;move index
      MOV   A,R1
      ANL   A,#RXBUFSZ-1       ;RXBUFSZ = 8-1 =7 0 through 7 = 8
      ADD   A,#RXBUF           ;# of characters in buffer+buffer address
      MOV   R0,A               ;R0 has buffer location
      MOV   A,@R0              ;get chr from buffer
      MOV   R1,A               ;put chr in R1
GCXIT:
      POP   PSW                ;restore previous state
      MOV   EA,C               ;restore previous interrupt state
      RET                      ;return
;
;*********************************************************************
SERINIT:                       ;Initialize serial port
      CLR   A
      MOV   TX_IN,A            ;initialize indexes
      MOV   TX_OUT,A
      MOV   RX_IN,A
      MOV   RX_OUT,A
      CLR   TR1
      CLR   TI
      CLR   RI
      MOV   SCON,#01011010B    ;TI SET INDICATES TRANSMITTER READY.
                               ;MODE 1 REN
      MOV   TMOD,#00100001B    ;TIMER 1 MODE 8 BIT AUTO RELOAD
```

```
      MOV   TH1,#0FDh           ;TIMER RELOAD VALUE
      SETB  TR1                 ;START TIMER
      SETB  NEEDTI
      SETB  ES
      RET
;
SINIT:
      DB    CR,LF
      DB    'Serial Port Initialized! '
CRLF: DB    CR,LF,0
;*******************************************************************
SER_ISR:
      PUSH  ACC                 ;store current state
      PUSH  PSW
      MOV   PSW,#00h
      PUSH  00h
      JNB   RI,TXISR            ;if not receive then check xmit
      CLR   RI                  ;else clear RI and process
      CLR   C
      MOV   A,RX_IN
      SUBB  A,RX_OUT            ;how many characters in buffer?
      ANL   A,#0-RXBUFSZ        ;
      JNZ   TXISR               ;jump if no characters in buffer
      MOV   A,RX_IN             ;else
      ANL   A,#RXBUFSZ-1        ;number of characters in buffer
      ADD   A,#RXBUF            ;plus buffer address =
      MOV   R0,A                ;buffer position
      MOV   @R0,SBUF            ;put received character in buffer
      INC   RX_IN               ;increment buffer position
;
TXISR:
      JNB   TI,ISRXIT           ;if TI=0 then exit
      CLR   TI                  ;else clear TI and process
      MOV   A,TX_IN             ;how many characters in buffer?
      XRL   A,TX_OUT
      JZ    TXBUFZ              ;jump if buffer empty
      MOV   A,TX_OUT            ;else
      ANL   A,#TXBUFSZ-1        ;number of characters in buffer
      ADD   A,#TXBUF            ;plus buffer address =
      MOV   R0,A                ;buffer position
      MOV   A,@R0               ;get character from buffer
      MOV   SBUF,A              ;send character out
      INC   TX_OUT              ;increment buffer position
      CLR   NEEDTI              ;need to set TI
      AJMP  ISRXIT              ;exit ISR
TXBUFZ:
      SETB  NEEDTI              ;buffer was empty dont need to set TI
ISRXIT:
      POP   00h                 ;restore and return
      POP   PSW
      POP   ACC
      RETI
;*******************************************************************
;*******************************************************************
```

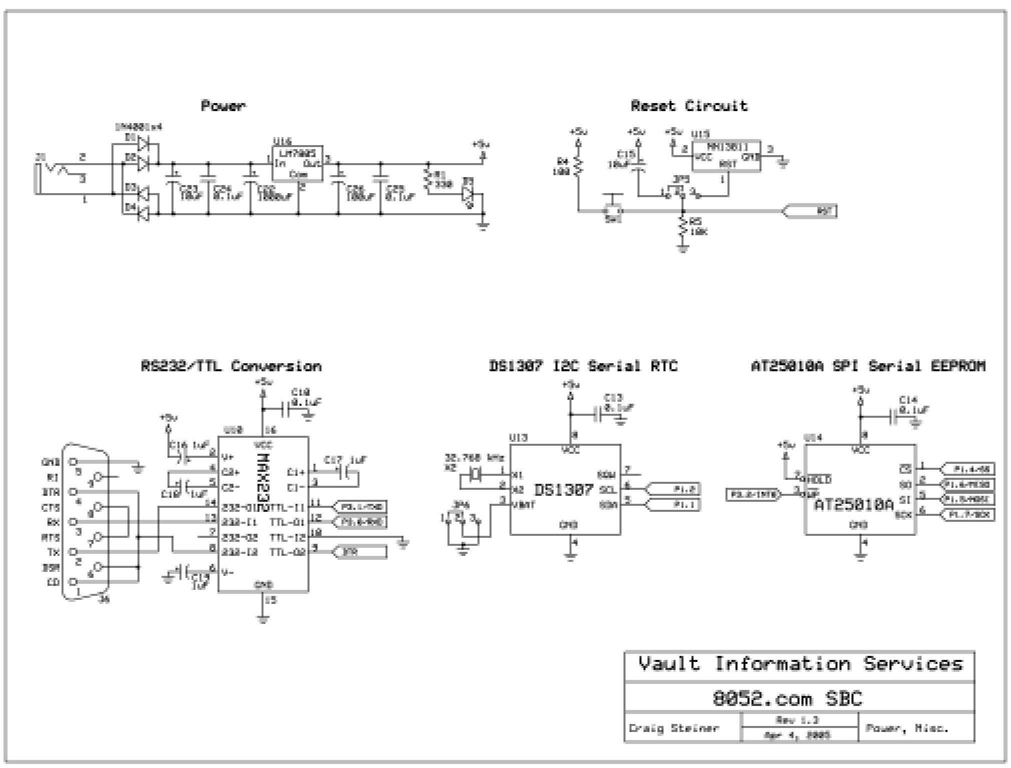
In conclusion

At this point I have gone way farther than I intended to in the first place. The above instructions and examples should get you started in serial communications with your micro controller project. Through further searching and reading on 8052.com and the internet, you should be able to find the answers to any other questions that may come up while implementing your serial application. I hope that this guide has helped you to get a working connection established and simplify the process of finding the information needed to do so.

Appendix



These two examples use a standard MAX232



Chapter 8 - Serial Port Operation

One of the 8051s many powerful features is its integrated *UART*, otherwise known as a 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.

Setting the Serial Port Mode

The first thing 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.
First, lets 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	Multiprocessor 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 / 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. Thus, well go through each bit and review its function.
The first four bits (bits 4 through 7) are configuration bits.
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. Well talk more about the various Serial Modes in a

moment.

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.

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.

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 data bits will be written to the serial line followed by a "set" ninth bit. If TB8 is clear the ninth bit will be "clear."

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 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.

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.

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 oscillators 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 11.059Mhz, 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 11.059Mhz 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.

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.

The following tables are for reference to the standard pin names and functions. For the basic communications shown in the previous examples you are only concerned with the TXD, TXD and SG or Ground. For all but the most demanding applications, the rest of the pins can be ignored.

Serial Pin outs (D25 and D9 Connectors)

D-Type-25 Pin No.	D-Type-9 Pin No.	Abbreviation	Full Name
Pin 2	Pin 3	TXD	Transmit Data
Pin 3	Pin 2	RXD	Receive Data
Pin 4	Pin 7	RTS	Request to Send
Pin 5	Pin 8	CTS	Clear to Send
Pin 6	Pin 6	DSR	Data Set Ready
Pin 7	Pin 5	SG	Signal Ground
Pin 8	Pin 1	CD	Carrier Detect
Pin 20	Pin 4	DTR	Data Terminal Ready
Pin 22	Pin 9	RI	Ring Indicator

PC = DTE (Data Terminal Equipment)
Modem = 8051 = DCE (Data Communications Equipment)

Pin Functions

The descriptions here reflect how the PC sees these signals.

Abbreviation	Full Name	Function
TXD	Transmit Data	Serial Data Output (TXD) - Data flowing from PC to modem
RXD	Receive Data	Serial Data Input (RXD) - Data flowing from modem to PC
CTS	Clear to Send	This line indicates to PC that the modem is ready to exchange data.
DCD	Data Carrier Detect	When the modem detects a "Carrier" from the modem at the other end of the phone line, this Line becomes active.
DSR	Data Set Ready	This tells the PC that the modem is ready to establish a link.
DTR	Data Terminal Ready	This is the opposite to DSR. This tells the modem that the PC is ready to link.
RTS	Request to Send	This line informs the modem that the PC is ready to exchange data.
RI	Ring Indicator	Goes active when modem detects a ringing signal from the PSTN.

Notes about USB

With more and more PC's and laptops coming out with no serial ports, only USB ports, it will be necessary for you to purchase a RS232/USB adapter/converter. There are many brands and choices.

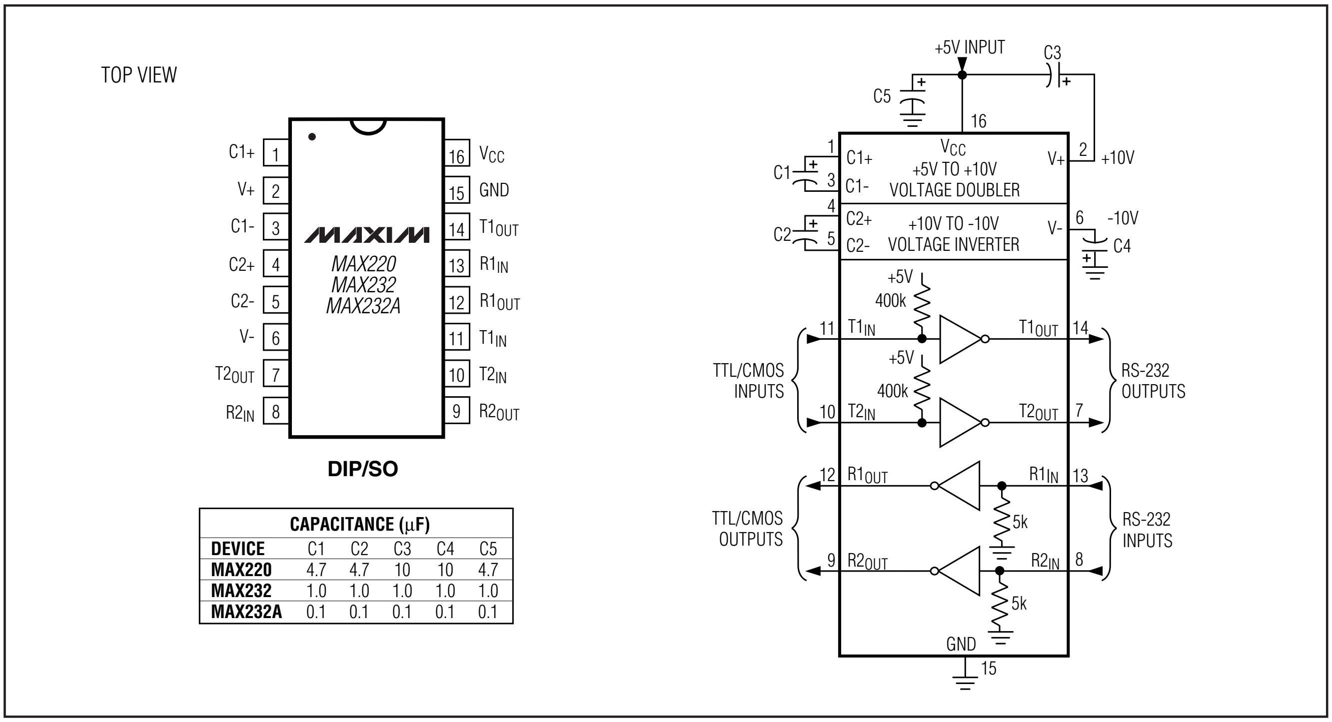
I haven't had to compare any of them, but I have a Targus model PA088 that I've had no problems with. Setting it up for a terminal program was as simple as installing the driver and plugging it in. It's recognized as a communications port, so you only have to select the correct port in the terminal program.

It is based on the following:

P87C52X2BBD - controller
PDIUSBD12 - USB
ADM211 - RS232
93LC66 - memory

It's one of the converters that Atmel recommends for FLIP, but I did have to patch FLIP for it to work correctly.

Beware of converters that are a **PL2303X**-based with a RS232-level converter labeled ZT213ECA. They were shown to have problems.



CAPACITANCE (μF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	4.7	4.7	10	10	4.7
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Figure 5. MAX220/MAX232/MAX232A Pin Configuration and Typical Operating Circuit

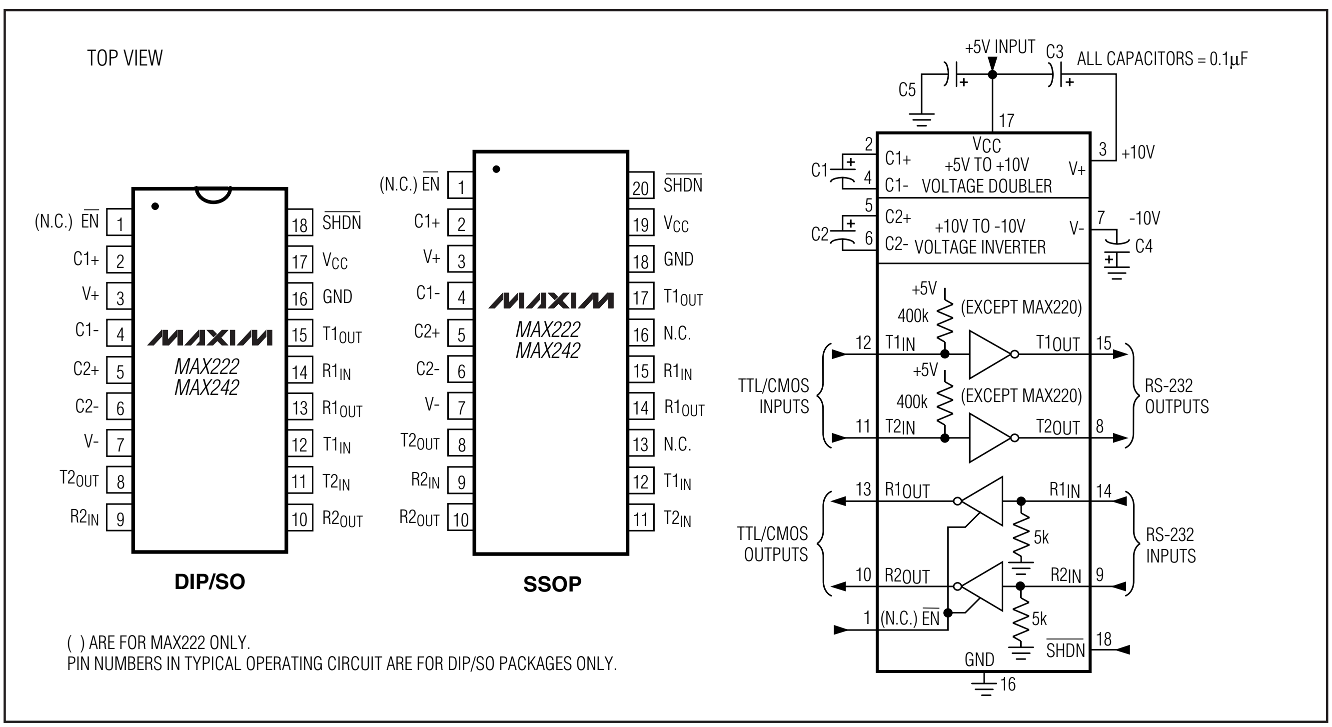


Figure 6. MAX222/MAX242 Pin Configurations and Typical Operating Circuit

Links

MAX232 http://pdfserv.maxim-ic.com/en/ds/MAX220-MAX249.pdf
(Note that in figure 11 on page 21 Pins 4 and 19 are labeled as RS232 Outputs when they should be RS232 Inputs.)

8052.com Tutorials http://www.8052.com/tutorial.phtml
(Complete tutorials on various topics concerning the 8051)

Chapter 8 “Serial Port Operation” http://www.8052.com/tutser.phtml
(This is included in the first part of the Appendix)

Craig Peacock’s Serial Guide http://www.beyondlogic.org/serial/serial.pdf
(This document is very good reading on serial communications)

Other useful Information http://www.beyondlogic.org

Keil’s Baud rate Calculator http://www.keil.com/c51/baudrate.asp
(Calculates timer reload values for particular crystal frequencies and desired baud rates)

The Final Word on the 8051
http://www.mcu-memory.com/mcu-book/THE_FINAL_WORD_ON_THE_8051.pdf
(How to use C on the 8051 lots of Keil examples)

Telecommunications Industry Association http://www.tiaonline.org
(This is where you can buy the RS232 standard specifications)

Reference Threads:

http://www.8052.com/forum/read.phtml?id=104713
http://www.8052.com/forum/read.phtml?id=104715
http://www.8052.com/forumchat/read.phtml?id=105092
http://www.8052.com/forum/read.phtml?id=57912
http://www.8052.com/forum/read.phtml?id=57912
http://www.8052.com/forum/read.phtml?id=46426
http://www.8052.com/forum/read.phtml?id=24916

http://www.8052.com/forum/read.phtml?id=73532
http://www.8052.com/forum/read.phtml?id=19395
(Interesting threads on polled vs. interrupt driven serial I/O)
http://www.8052.com/forum/read.phtml?id=66097
(Thread concerning flow control - not real useful)
http://www.8052.com/forum/read.phtml?id=23266
(Circular buffers by pointers or index)