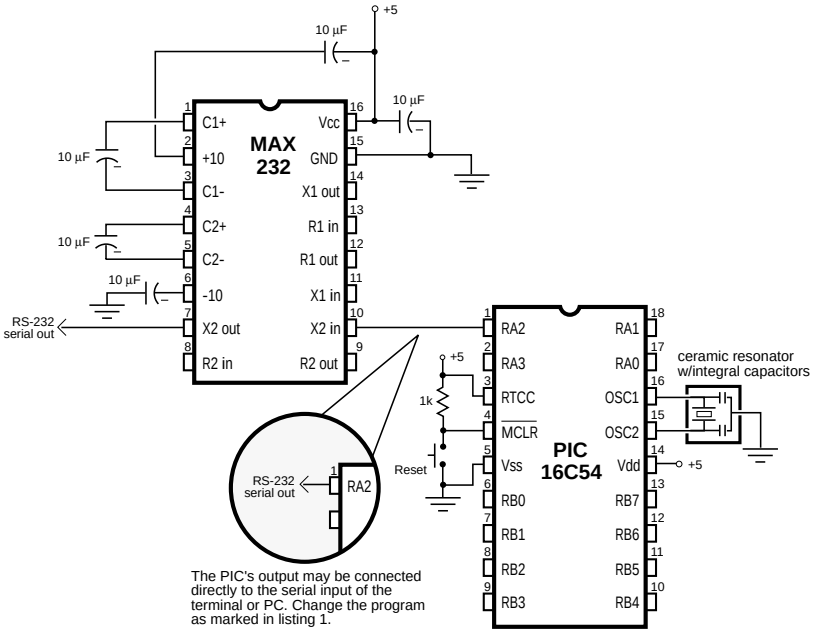
Sending RS-232 Serial Data

Introduction. This application note covers transmission of asynchronous serial data using PIC microcontrollers. It presents an example program in TechTools assembly language that transmits a text string serially via RS-232 at speeds up to 19,200 bits per second. Hardware examples demonstrate the use of a popular serial line driver, as well as a method for using the PIC's output to directly drive a serial line.

Background. Many PIC applications require sending data to a larger computer system. The common RS-232 serial port is almost ideal for this communication. While the PIC lacks the onboard serial communication hardware available with some more expensive controllers, it can readily be programmed to add this capability.

A previous application note (#2, Receiving RS-232 Serial Data) covered the data format and RS-232 signals in some detail. Here are the highlights:

A byte of serial data is commonly sent as 10 bits: a start bit, eight data



The PIC's output may be connected directly to the serial input of the terminal or PC. Change the program as marked in listing 1.

bits, and a stop bit. The duration of these bits is calculated by dividing the rate in bits per second (commonly baud) into 1 second. The stop bit can be stretched to any desired duration.

Under the RS-232 standards, a high (1) is represented by a negative voltage in the range of -5 to -15 volts. A low (0) is a voltage from +5 to +15 volts. In addition to connections for data input, output, and ground, most RS-232 ports include handshaking lines that help devices turn the flow of data on and off when necessary (for instance, when a printer is receiving data faster than it can store and process it). Many applications avoid the use of these hardware signals, instead embedding flow-control commands in the data stream itself.

A traditional method of sending serial data is to use a parallel-in serial-out shift register and a precise clock. The start, data, and stop bits are loaded into the register in parallel, and then shifted out serially with each tick of the bit-rate clock. You can easily use a software version of this method in a PIC program. Really, the most difficult part of transmitting an RS-232-compatible signal from the PIC is achieving the signaling voltages. However, the hardware example presented here shows that:

- There are devices that generate the RS-232 voltages from a single-ended 5-volt supply.

- It is possible to use the PIC's output to directly drive an RS-232 input, if the cable is kept short.

How it works. The PIC program in listing 1 sends a short text string as a serial data stream whenever the PIC is reset. To reset the PIC, push and release the reset button shown in the schematic . If you lack appropriate terminal software for your computer, listing 2 is a BASIC program that will help get you started. If you use other terminal software and get "device timeout" errors, try cross-connecting the handshaking lines as shown in figure 2 below. This has the effect of disabling handshaking. The program in listing 2 does this in software.

The PIC program starts by setting port RA to output. Serial data will be transmitted through pin RA.2. The program consists of two major loops, defined by the labels *:again* and *:xmit*. The first loop, *:again*, initializes the

bit counter, and then gets a byte from the subroutine/table called *string*. Once *string* returns a byte in the *w* register, the program puts this byte into *xmt_byte*. After a start bit is sent by clearing the serial-out pin, *:xmit* takes over. It performs the following steps:

- Rotate the transmit byte right (into carry).
- Move the carry bit to serial output.
- Wait one bit time (set by *bit_K*).
- Decrement the bit counter.
- Is the counter zero?
  - > No, loop again.
  - > Yes, exit the loop.

With the *:xmit* loop finished, the program sets the serial-out pin to send a stop bit, and checks to see whether it has reached the end of the string. If not, it goes back to *:again* to retrieve another byte. If it is done with the string, the program enters an endless loop.

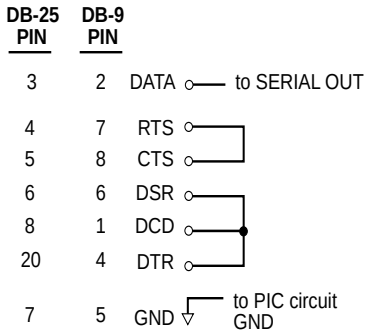


Figure 2. Hookups for standard 9- and 28-pin connectors. Connecting RTS to CTS disables normal handshaking, which is not used here.

The hardware portion of the application is fairly straightforward. The PIC produces logic-level signals that the MAX232 chip converts to acceptable RS-232 levels, approximately ±10 volts. The MAX232 has onboard charge-pumps that use the external capacitors to build up the required

signaling voltages. There are other chips that provide additional features, such as additional I/O channels, higher signaling rates, lower power consumption, and the ability to work without external capacitors.

Modifications. The MAX232 draws more current and costs more (in single-part quantities) than the PIC it supports. Most of the time, this is still a bargain compared to adding a bipolarity power supply to a device just to support RS-232 signaling. In fact, the MAX232 has excess current capacity that can be used to power other small bipolarity devices, such as low-power op-amps. Depending on the application, additional voltage regulation may be required.

If you plan to use the serial port infrequently, consider powering the MAX chip through one of the PIC's I/O pins. Your program could turn the MAX on shortly before it needed to send a serial message, and turn it back off afterward. This option is especially attractive for uses that require wringing the most life out of a set of batteries.

A sample MAX232 that we tested drew 15 mA while driving two simulated serial-port loads. This is well within the PIC's drive capability. A further test showed that the MAX232's output voltages rose to their full ±10-volt levels about 1.5 milliseconds after power was applied to the chip. If your program switches the MAX232 on and off, program a 1.5-ms delay between turning the device on and sending the first bit.

In some cases, you may be able to dispense with the serial line driver altogether. Make the changes marked in listing 1 "for direct connection" and wire RA.2 directly to the serial receive connection of your terminal or PC. The changes in the program invert the logic of the serial bits, so that a 1 is represented by an output of 0 volts and a 0 is a 5-volt output. According to RS-232, the receiving device should expect voltages of at least -3 volts for a 1 and +3 volts for a 0. Voltages lying between +3 and -3 are undefined, meaning they could go either way and still comply with the standard.

As a practical matter, though, it wouldn't be smart to let 0 volts be interpreted as a 0, since this is a serial start bit. Any time the serial input was at ground potential, the terminal or PC would attempt to receive incoming serial data. This would cause plenty of false receptions. Serial-

port designers apparently take this into account and set the 0 threshold somewhere above ground.

That's why this cheap trick works. Don't count on it to provide full RS-232 performance, especially when it comes to sending data rapidly over long cables. Keep cables short and you shouldn't have any problems (19,200 baud seems to work error-free through 6 feet of twisted pair).

If your application will have access to the handshaking pins, you may be able to steal enough power from them to eliminate batteries entirely. According to the RS-232 specifications, all signals must be capable of operating into a 3000-ohm load. Since many computers use ±12 volts for their RS-232 signals, each line should be capable of delivering 4 mA. In practice, most can provide more, up to perhaps 15 mA. The only problem with exploiting this free power is that the software running on the PC or terminal must be written or modified to keep the handshaking lines in the required state.

One final hardware note: Although timing isn't overly critical for transmitting serial data, resistor/capacitor timing circuits are inadequate. The PIC's RC clock is specified to fairly loose tolerances (up to ±28 percent) from one unit to another. The values of common resistors and capacitors can vary substantially from their marked values, and can change with temperature and humidity. Always use a ceramic resonator or crystal in applications involving serial communication.

Program listing. This program may be downloaded from our Internet ftp site at ftp.tech-tools.com. The ftp site may be accessed directly or through our web site at http://www.tech-tools.com.

Values of Timing Constant Bit_K for Various Clock Speeds and Bit Rates

Clock Frequency	Serial Bit Rate (bit time)						
	300 (3.33 ms)	600 (1.66 ms)	1200 (833 µs)	2400 (417 µs)	4800 (208 µs)	9600 (104 µs)	19,200 (52 µs)
1 MHz	206	102	50	24	Ñ	Ñ	Ñ
2 MHz	Ñ	206	102	50	24	Ñ	Ñ
4 MHz	Ñ	Ñ	206	102	50	24	Ñ
8 MHz	Ñ	Ñ	Ñ	206	102	50	24

Other combinations of clock speed and bit rate can be supported by changing the bit_delay and start_delay subroutines. The required bit delay is $\frac{1}{bitRate}$. For example, at 1200 baud the bit delay is $\frac{1}{1200}$ = 833µs. The start delay is half of the bit delay; 416µs for the 1200-baud example. Calculate the time delay of a subroutine by adding up its instruction cycles and multiplying by $\frac{4}{clockSpeed}$. At 2 MHz, the time per instruction cycle is $\frac{4}{2,000,000}$ = 2µs.

```
; PROGRAM: XMIT232
; This program transmits a string of serial data. The baud rate is determined
; by the value of the constant bit_K and the clock speed of the PIC. See the
; table in the application note (above) for values of bit_K. For example, with
; the clock running at 4 MHz and a desired transmitting rate of 4800 baud,
; make bit_K 50.

; >>> Remember to change device info if programming a different PIC! <<<
; Do not use RC devices. Their clock speed is not sufficiently accurate
; or stable enough for serial communication.

          device    pic16c54,xt_osc,wdt_off,protect_off
          reset     begin

bit_K     =         24              ;24 for 19,200-baud operation @ 8 MHz
serial_out=         ra.2

          org  8                    ;Start of available RAM

delay_cntr          ds        1     ;Counter for serial delay routines
bit_cntr            ds        1     ;Number of transmitted bits
msg_cntr            ds        1     ;Offset in string
xmt_byte            ds        1     ;The transmitted byte

          org  0                    ;Start of code space (ROM)

begin     mov       !ra, #00000000b ;Set port to output.
```

```
            mov     msg_cntr, #0        ; Message string has ten
                                        ; characters 0 through 9.

:again      mov     bit_cntr,#8         ;Eight bits in a byte.
            mov     w,msg_cntr          ;Point to position in the string.
            call    string              ;Get the next character from string.
            mov     xmt_byte,w          ;Put character into the transmit byte.
            clrb    serial_out    ;Change to setb serial_out for direct connection.
            call    bit_delay           ;Start bit.

:xmit       rr      xmt_byte            ;Rotate right moves data bits into
                                        ;carry, starting with bit 0.
            movb    serial_out,c        ;Change to movb serial_out,/c for
                                        ;direct connection.
            call    bit_delay           ;Data bit.
            djnz    bit_cntr,:xmit      ;Not eight bits yet? Send next data bit
            setb    serial_out          ;Change to clrb serial_out
                                        ;for direct connection
            call    bit_delay           ;Stop bit.
            inc     msg_cntr            ;Add one to the string pointer.
            cjbe    msg_cntr,#9,:again  ;More characters to send? Go to the top

:endless    jmp     :endless            ;Endless loop.
                                        ;Reset controller to run program.

; To change the baud rate, substitute a new value for bit_K at the beginning of
; this program.

bit_delay   mov     delay_cntr,#bit_K
:loop       nop
            djnz    delay_cntr, :loop
            ret

string      jmp     pc+w                ;String consisting of "TechTools"
                                        ;followed by a linefeed.
            retw    'T','e','c','h','T','o','o','l','s',10
```

BASIC Program for Receiving Text String via COM1

```
10 CLS
15 REM Substitute desired baud rate for 19200 in the line below.
20 OPEN “com1:19200,N,8,1,CD0,CS0,DS0,OP0” FOR INPUT AS #1
30 IF NOT EOF(1) THEN GOSUB 200
40 GOTO 30
200 Serial$ = INPUT$(LOC(1), #1)
210 PRINT Serial$;
220 RETURN
```