

FEATURE ARTICLE

Tom Napier

An Intelligent Serial Command Interface

Specializing has its perks. If a device is not available commercially today, the opening is there for the taking. And, sometimes the development is even fun. Tom knows that this is part of the joy of being a consultant, as he runs through the process of designing a box to speed up the testing of communications equipment.



One of the fun things about being a consultant is that you get asked to supply devices that are not commercially available. These are often not inherently difficult to develop, but they have such a specialized function that no company has considered manufacturing them.

I was recently asked to design a box to speed up the testing of telecommunications equipment. It needed to allow the computer running the tests to remotely switch either of two input signal lines to any of four inputs on the unit being tested. The box I designed to perform this task has unusual input and output connectors switched by relays (nothing particularly interesting there). However, the relays are switched by an 18-pin PIC microcontroller programmed as a smart modem.

THE SMART MODEM

The modem sits on a 9600-bps serial line that is transmitting data and

control signals to other equipment. It reacts only to a specific pattern of characters and interprets them as relay switching commands. Up to seven devices can be coupled to the same serial line and controlled individually or globally. Luckily no transmit function was required, so there was no signal contention to consider.

The input impedance is 3.3 kilohms, the RS-232 standard. The serial input is a conventional 9-pin D connector wired in parallel with the other equipment, and if there had been panel space, I would have added a second connector to allow pass-through wiring of the signal. Fortunately, the transmitting computer is able to drive several RS-232 inputs in parallel.

A valid command consists of six printable ASCII characters, of which the first is a colon, the last is a semicolon, the second is the device address, and the remaining three are relay commands. Obviously this format can easily be extended. Each character has one start bit, eight data bits, and one stop bit, but the eighth (parity) bit is ignored.

The address character is any letter from A to G. Internal jumper plugs set the address of each device from 1 through 7. If a device is set to address 0 it responds to all command strings regardless of the address. The same command can be sent to all connected devices by using an "@" as the address character.

In the original device, all three command characters lay in the range of 0 to 4. Numbers 1 through 4 selected one of four output relays, and 0 turned all the relays off. Again, this could easily be expanded. Sending 0 through 3 as the third command character sent two TTL-level signals to an auxiliary connector to switch other equipment. Sending a 4 in this position reset the box to its default condition.

A typical command string such as :C220; switched both inputs of device

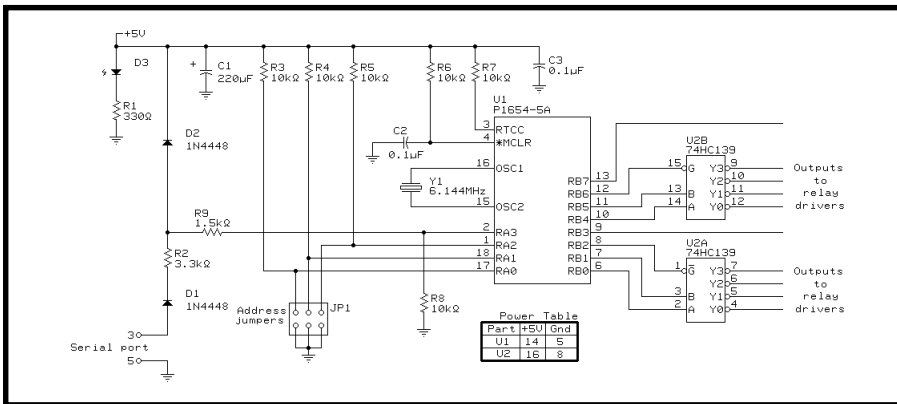


Figure 1—This partial schematic shows the vital parts of the remote control receiver.

C to their respective output 2 and left the auxiliary outputs high. The string :@114; reset all connected devices to their default condition. The first two digits don't matter in this case.

THE HARDWARE

The important part of the device consists of the 16C54, a 74HC139 decoder chip, and a bunch of SN75453 driver chips. The serial input interface uses three resistors and two diodes to match the RS-232 input to the PIC's input levels. One pin of the PIC's 4-bit port is the serial input, and the other three pins of this port go to the jumpers, which select the device address. Six pins of the 8-bit port drive the two halves of the decoder chip, and the remaining two pins control the auxiliary drivers. Figure 1 shows the essential parts.

The remote control box is powered from a 9-V 300-mA AC/DC adapter. A diode protects it from a reversed polarity input, and a 5-V regulator supplies power to the microcontroller, drive logic, and relays. A green LED indicates that power is connected to the device, and LEDs across each output relay coil show which output is active.

THE FIRMWARE

The PIC16C54 microcontroller runs at a clock rate of 6.144 MHz. The serial input algorithm was conservatively designed so that it would not be confused if it powered up in the middle of a continuous string of irrelevant ASCII characters. It bit-slips through successive random characters until it locks on the start and stop bits. Then it looks for a colon. All other characters

are ignored. (The command strings are transmitted as comments so that the other equipment being controlled does not get confused by them. Colons are not used to control the other equipment in the system.)

After finding a colon, the microcontroller starts examining successive characters. If the next character is neither the assigned address character or an "@", the unit switches back to search mode. Otherwise, it starts storing characters and comparing them to the valid range. Any invalid character immediately restarts the colon search. Even the right number of valid characters must be followed by a semicolon before anything further happens. At that point, the selected commands are sent to the relay driver chips via the PIC's output pins.

Whenever a command switches an input from one output connector to another, the firmware turns off the current output relay. It waits long enough for the relay to switch off before turning on the new output. This avoids overloading the signal source. However, the program first checks whether a command will change the output relays. This stops it from switching an output off momentarily (i.e., it avoids output glitches) if the same command is sent twice or if the auxiliary outputs are switched without altering the output relay connection. Listing 1 shows the PIC program.

A UNIVERSAL REMOTE CONTROL

The outcome of this development is a compact remote control receiver, which can be adapted to many other functions with a few changes to the

firmware. System size and cost have been reduced by combining the serial interface and the function decoding in a single chip. ■

Although Tom Napier is an analog electronics consultant, he continues to find excuses to do interesting things with microcontrollers. He hopes you will find them interesting, too.

SOFTWARE

The software is available for downloading on the *Circuit Cellar* web site.

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, subscribe@circuitchellar.com or www.circuitchellar.com/subscribe.htm.

Listing 1

```
; UART/Control program for a signal switcher
; Version 2.0, 13 December 1999

; COPYRIGHT (C) 1999 T. M. Napier.

; Written for 6.144 MHz clock, 651 nS per instruction

; Input/Output
;   Port A all inputs
;   Bit 0   Device address 0
;   Bit 1   Device address 1
;   Bit 2   Device address 2
;   Bit 3   Serial port input

;   Port B all outputs
;   Bit 0   Bank B select 0
;   Bit 1   Bank B select 1
;   Bit 2   Bank B enable, 0 = enabled
;   Bit 3   Auxiliary bit 0

;   Bit 4   Bank A select 0
;   Bit 5   Bank A select 1
;   Bit 6   Bank A enable, 0 = enabled
;   Bit 7   Auxiliary bit 1

;Function registers

INDEX EQU 0
RTCC EQU 1
IREG EQU 4
PORTA EQU 5
PORTB EQU 6

;System registers

TEMP EQU 8 ;Temporary register
TIMER EQU 9 ;Timer count for ASCII I/O
PARAM EQU 10 ;Subroutine parameter register
OLD EQU 11 ;Current relay status
NEW EQU 12 ;Used to build new relay status
TIME1 EQU 13 ;Used to implement delay
TIME2 EQU 14

STRO EQU 16 ;Start of string buffer
STR1 EQU 17
STR2 EQU 18

ENDIT EQU 0F3H ;Terminal value of IREG, 0E0H+13H
;Remember, top 3 bits of IREG read as ones!

;PORT A serial input bit

SDAT EQU 3

;PORT B bits

DISB EQU 2 ;High = relay B disable
AUX0 EQU 3
DISA EQU 6 ;High = relay A disable
AUX1 EQU 7

;Serial port timing

WHOLE EQU 20 ;Timer counts per bit
HALF EQU 10 ;Timer count for half a bit
BYTE EQU 190 ;9.5 bit periods of ASCII input byte

;Miscellaneous

COLON EQU 3AH ;Start character
SEMI EQU 3BH ;Terminator
DEFAL EQU 8BH ;Default condition
; Both channels set to 1, auxiliary relay open

GOTO START ;skip past subroutines
```

Listing 1—continued

```

; -----
;Wait for ASCII input, put it in PARAM.
;Checks for valid start bit and stop bit.
;Input low = 1, high = 0, start bit = high.
;RTCC controls the bit sampling time.
;Uses TIMER as bit time reference.

ASCIN    BTFSC    PORTA,SDAT    ;Test serial input
         GOTO     ASCIN        ;Wait for stop bit

ASCIO    BTFSS    PORTA,SDAT    ;Test serial input
         GOTO     ASCIO        ;Wait for start bit

         CLRF     RTCC         ;Clear timer
         MOVLW   HALF         ;Half a bit period
         MOVWF   TIMER        ;Set up next sample time

ASC11    MOVF     RTCC,0
         SUBWF   TIMER,0
         SKPZ
         GOTO     ASC11        ;Delay half a bit

         BTFSS   PORTA,SDAT    ;Check start bit is still high
         GOTO     ASCIN        ;Else try again

ASC12    MOVLW   WHOLE         ;One bit period
         ADDWF   TIMER,1      ;Update sample time target

ASC13    MOVF     RTCC,0
         SUBWF   TIMER,0
         SKPZ
         GOTO     ASC13        ;Wait until next bit time

         MOVLW   BYTE
         SUBWF   TIMER,0      ;Check if done
         SKPNZ
         GOTO     ASC14        ;Got to stop bit so exit

         RRF     PARAM,1      ;Shift byte store
         BCF     PARAM,7      ;Put a zero in MSB
         BTFSS   PORTA,SDAT    ;Skip if input is high (=0)
         BSF     PARAM,7      ;Put a one in MSB
         GOTO     ASCI2        ;Wait for next bit

ASC14    BTFSC    PORTA,SDAT    ;Check for stop bit (low)
         GOTO     ASCIN        ;Else abandon character

         BCF     PARAM,7      ;Clear parity bit
         RETLW   0

; -----

;Number in W is wait time in 0.5 mS units

WAIT     MOVWF   TIME2        ;Set time delay
         CLRF   TIME1
WT        DECFSZ  TIME1,1
         GOTO   WT            ;1.953 uS per loop

         DECFSZ  TIME2,1
         GOTO   WT            ;500 uS per loop

         RETLW   0

; -----

;Start of program

START    MOVLW   -1
         TRIS   5            ;Set port A to inputs
         MOVLW   2
         OPTION DEFAL        ;Set data rate to 9600 baud
         MOVLW   DEFAL

```