

# Audio/Video Traffic Control

## FROM THE BENCH

Jeff Bachiochi



When Steve announced his plans for a new "entertainment room," I knew there was bound to be a flurry of new equipment purchases. He knows what he wants and if he can't find the right piece of equipment, he'll build it. The day Steve divulged his intentions for this project, I had mixed feelings.

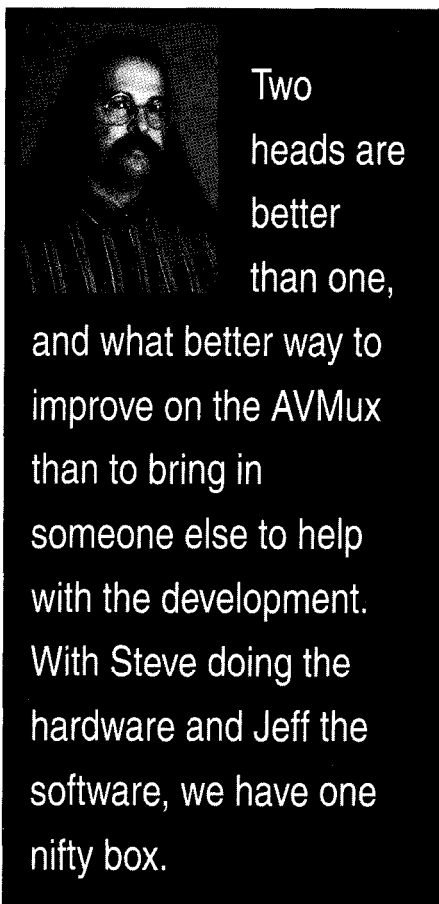
You see, I've been meaning to put together a video switcher since I built our addition three years ago. At that time, I installed a coax line to each of the rooms. I figured this would eventually allow me to choose from various video sources from each remote location. If I didn't have a hand in this project, I may not be getting the features I was looking for. On the other hand, if I collaborated, it could end up being a better project all around.

By the time Steve and I had exchanged our wish lists, it was clear that we were going in two entirely different directions. Steve wanted independent audio and video switching capabilities. I was looking for remotely accessed video-only switching. Our modular design approach allows one to trim away unwanted functions

without having to redesign the entire system.

Although prototyped on the same PC board, the video and audio sections are actually separate designs and could be separate PC boards. With this approach, the multiplexer could include a video switcher, an audio switcher, or both.

As Steve previously explained, control of the multiplexer can come from a variety of sources. First, a serial terminal connection gives the user an access port for program maintenance. With the mode switch set in local mode, a matrix of video and/or audio connections is presented to the terminal, along with instruction prompts for changing the matrix connections. In remote mode, ASCII-coded matrix data is transmitted and is intended for use with the remote LED matrix display. The display indicates the present video and/or audio connections on an 8x8 LED array and prompts the user to provide additional commands using the attached keypad. A



Two heads are better than one,

and what better way to improve on the AVMux than to bring in someone else to help with the development. With Steve doing the hardware and Jeff the software, we have one nifty box.

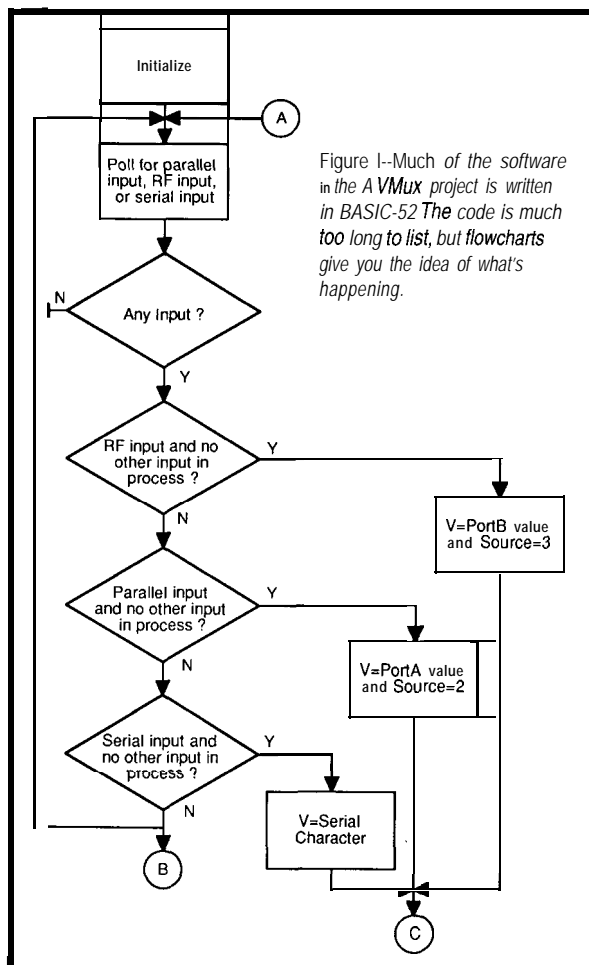


Figure 1--Much of the software in the AVMux project is written in BASIC-52. The code is much too long to list, but flowcharts give you the idea of what's happening.

second (auxiliary) serial output contains connection information displayed in a from/to listing using the actual names of equipment connected to the AVmux's I/O jacks.

Input can come from two additional sources (other than the serial input). An 8255 is configured as a pair of strobed parallel input ports. The first port is used like a printer port, accepting parallel data from a PC or DIO-Link. The second port interfaces to Steve's RF receiver (see his article on page 36 of this issue) to decode remote keypad pushbuttons.

Two configuration switches are used. The first, as stated earlier, selects a "local mode," where commands can arrive from a serial terminal, or an "LED matrix mode," where commands can arrive from the hand-held LED matrix and control terminal. The

second switch enables or disables the auxiliary serial output, which is meant for display through a video monitor.

### A/V SOFTWARE

The main code is written in BASIC-52 with a short assembly language call used to bit bang the array data into the video multiplexer and audio multiplexer. Both devices use a clocked-serial bit stream as input with a load pulse to make the final parallel transfer of data from the internal shift registers. If the audio multiplexer did not have a minimum clock rate of 20 kHz, I could have skipped the assembly stuff, but the need for speed mandated it. Parameters are passed to the routine through a few internal user registers prior to making the call. The routine grabs data from one of the two data tables held in protected memory

above MTOP. The data is transferred by the CALLED routine and shifted into both multiplexer devices. Depending on which device is the destination, the appropriate load strobe is pulsed.

The assembly language routine could have been permanently set in place, but I chose to embed the routine into the BASIC program as DATA statements and XBY (poke) it into protected memory when the BASIC program is run. This allows all the code to exist in a single easily maintained program. See Figure 1 for a gross overview of this program.

The BASIC code begins with a little house cleaning and then reads the DATA statements into protected memory. A few checks are made at this point. First, a checksum is calculated to ensure the DATA statements were read correctly. Next, the

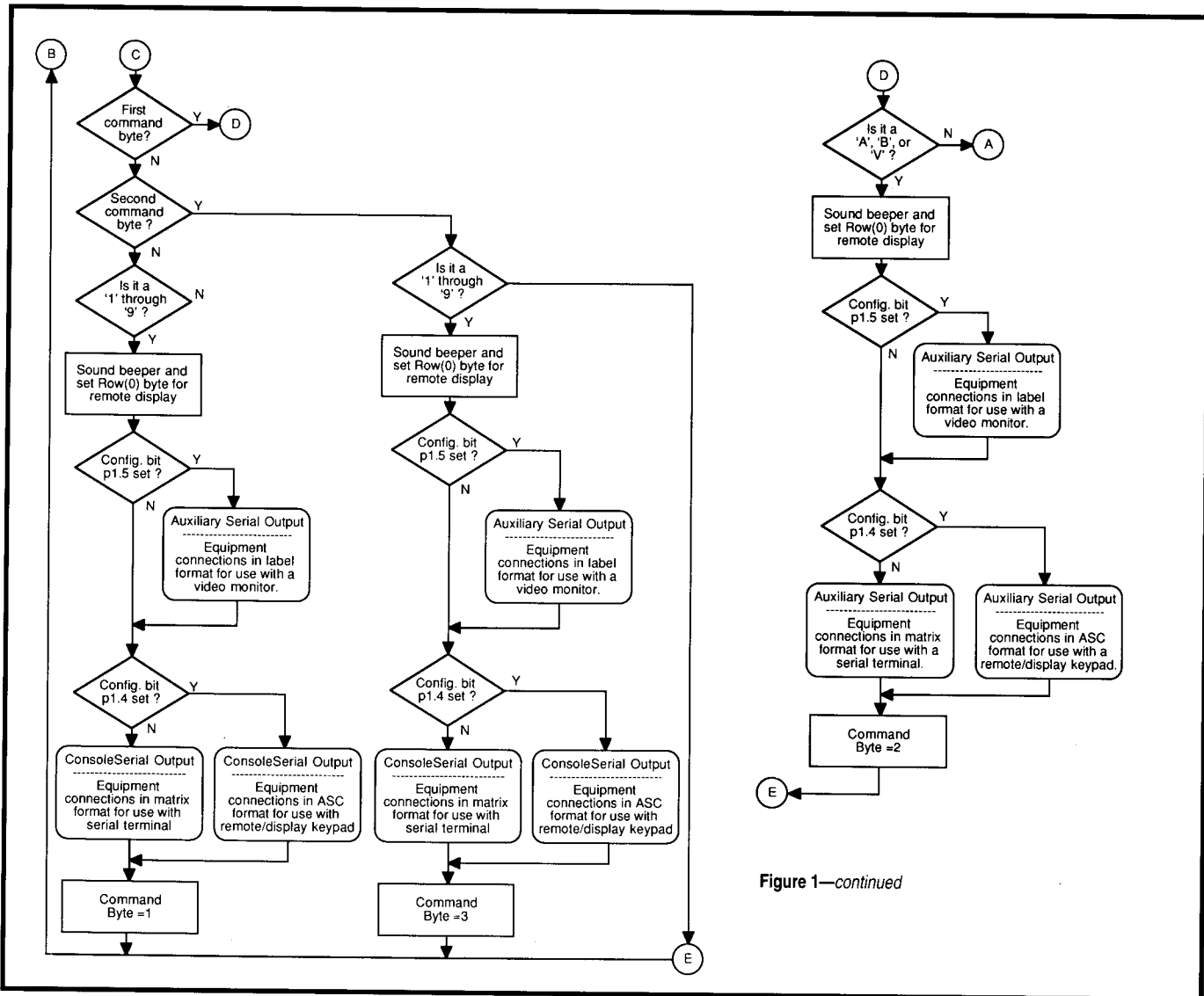


Figure 1—continued

program verifies that there was protected memory in the system to accept it. We wouldn't want to make a CALL without being sure we could get back.

Four tables are kept in protected memory just below the above routine. Table 1 has four bytes of video-array data. Each byte holds the connection data for two output channels, one in each nybble. Legal nybble values are 0-7 (connected to input channel 1-8) or 9 (which means a grounded input).

Table 2 holds 32 bytes of audio-array data. Each word (two bytes) holds a 16-bit data value that indicates whether any of the 16 inputs is connected to a particular output. A "1" in a particular bit position (bits 0-15 equate to inputs 1-16) signifies that that input is connected to the output associated with that output word (here 0-15 means outputs 1-16). Because the audio matrix is actually 16x16 and we want to use it as a dual [stereo] 8x8 matrix, the matrix is divided up into

four pieces: X1-8 (left inputs), X9-16 (right inputs), Y 1-8 (left outputs), and Y9-16 (right outputs). Notice that some connections are considered illegal even though they can be logically made (i.e., left input 1 is not allowed to be connected to right output 1).

Table 3 is a list of labels for each of the 16 video and 16 audio I/O connections (24 characters maximum each). I choose not to prompt for these labels from within this program. But instead, since you would have to connect a terminal device up to enter the data anyway, I created a short program containing the labels (which you edit directly in the program before downloading it). When the program is run, your labels are inserted into protected memory. This program can be easily edited and run if you ever make connection modifications.

Table 4 is a simple 2-byte flag that indicates if the program has been run

before. Once the program has been run, we don't want to initialize the A/V multiplexer with no connections, because in all likelihood we want to initialize all connections to their prior state before the power was removed [the last time it was on].

By checking Table 4, we know how to initialize the audio and video multiplexers. The data from Tables 1 and 2 are transferred using the CALL routine and we are now ready for user input. The first legal input is restricted to either an "A" (audio), a "B" (Both), or a "V" (video). This could come in through the serial port or through the 8255 PPI (recall that this is set up as two strobed input ports). The 8255 handles both parallel inputs, one from parallel port and one from the RF receiver. Once a legal character is received from one device, the others are locked out until the command is completed. This is fulfilled by receiving two additional characters in the

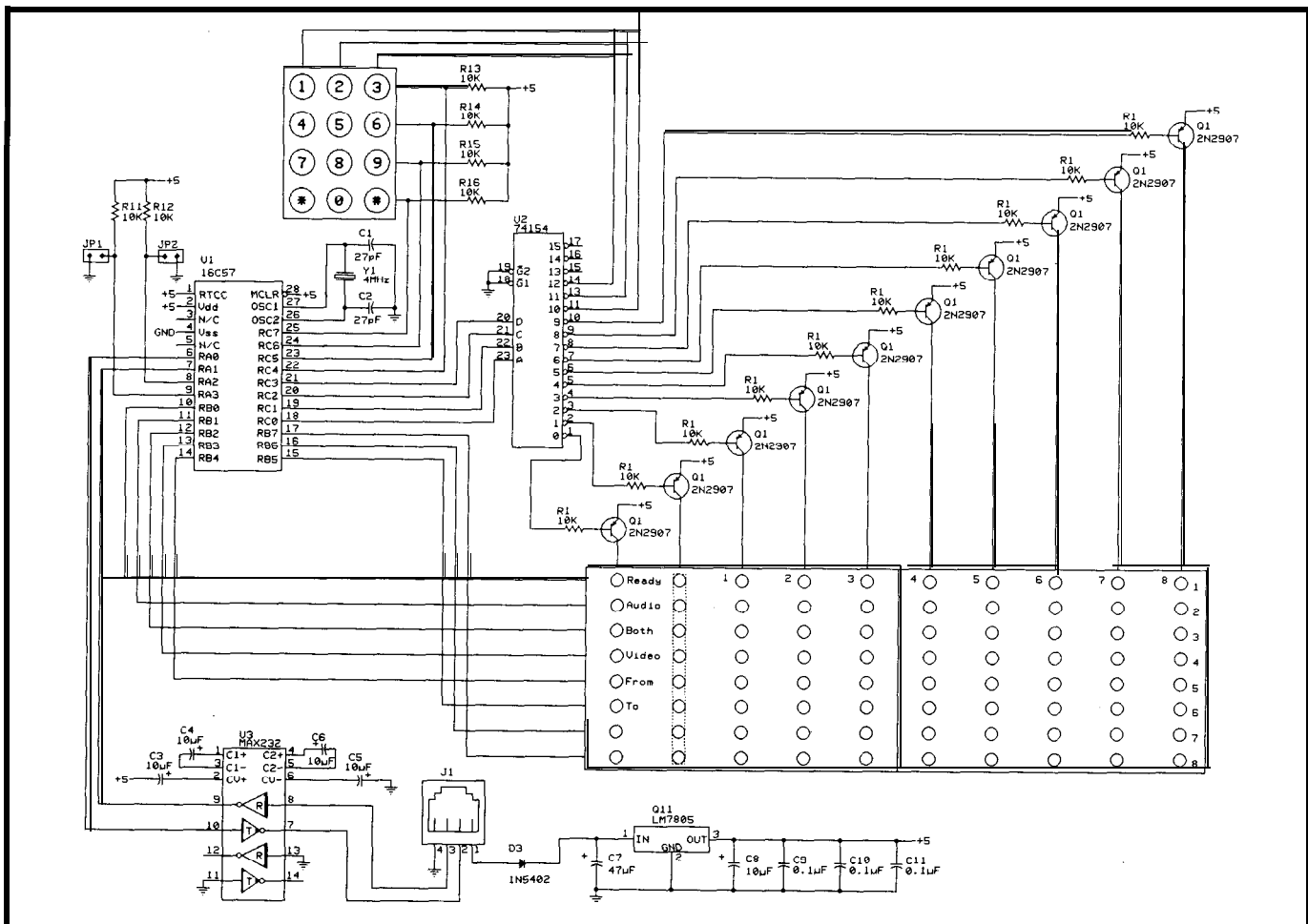


Figure 2-At the core of the AVmux remote is a PIC controller chip. To maximize the use of the PIC's I/O lines, the keypad scanning is multiplexed with the displays. A typical MAX232 is used to talk to the AVmux's 80C52 processor serially

range of "1"- "9." The first character selects where the audio or video signal is coming from, which can be input 1-8 or 9 (which is equivalent to none). The second character determines which output the input is assigned to, which can be output 1-8 or 9 (to cancel the command).

After the reception of each legal character, the configuration switches determine which BASIC subroutines are used to output the appropriate feedback to one or more of the display devices.

### REMOTE LED MATRIX AND KEY PAD

Perhaps you want to view the I/O connections selected for the audio and/or video multiplexer without the

Photo 1--The 8x8 LED matrix makes for an *easy-to-read* panel when trying to discern which and what type of I/O connections have been made.

hassles of a serial terminal or video monitor attached. An 8-column by 8-row matrix of LEDs will do nicely in this event. Each column represents an output or signal source while each row designates an input or component destination. A glowing LED at the intersection of a source column and a destination row designates a connection. See Photo 1.

Individual LEDs could be used, however I had a few 5x8 LED arrays left over from an earlier FTE project (the scrolling LED display). Two of these make a 10x8 matrix; the first

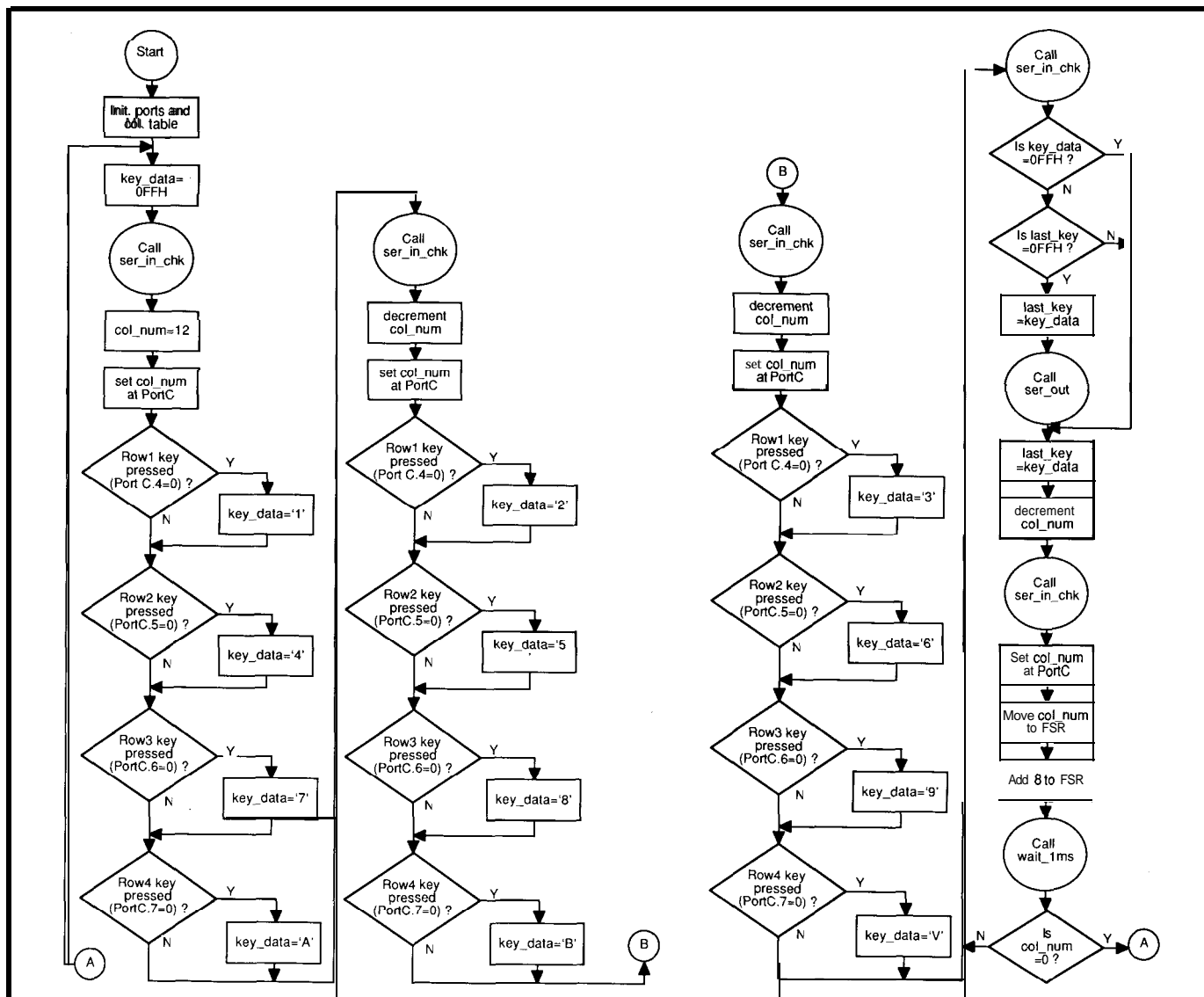
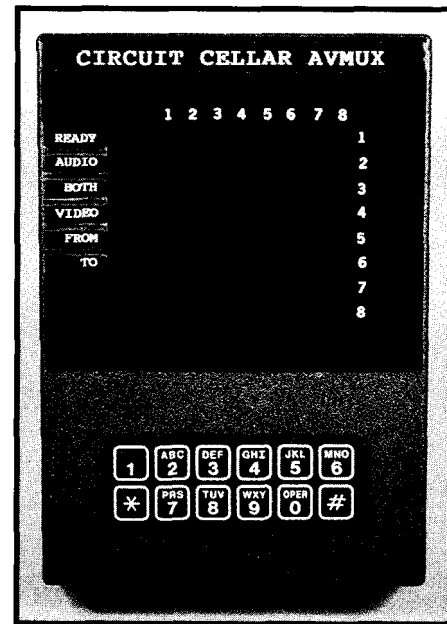


Figure 3—The PIC must simultaneously scan the keypad, scan the display, and watch for serial data from the AVMUX. The lack of interrupts necessitates lots of polling, which can make the code look more complicated than it is.

two columns are not used for matrix indicators, but for prompting input from the user. Six LEDs in the first row are labeled as Ready, Audio, Both, Video, From, and To. When a command is finished, the "Ready" LED prompts the user to choose from either the Audio, Video, or Both matrices. If entered correctly on the keypad, the "Audio," "Video," or "Both" LED comes on along with the prompting of the "From" LED. The 8x8 LED matrix is updated to show the present connections of the user's chosen matrix. When a column number is entered on the keypad by the user (1-8 or 9 as none), the "To" LED is turned on prompting the user for a final entry, which will be the destination. A row number completes the command (1-8 or 9 to cancel) and updates the 8x8 LED matrix with the new connection data.

You may wish to have this display out front with the rest of your component controls or closer at hand. Therefore, its design incorporates a serial transmission scheme. The umbilical cord, a modular phone cable, carries RS-232 transmissions using three conductors-transmit, receive, and ground-while the fourth conductor carries power for the I/O display. This allows the remote LED display unit to be a considerable distance from the switching electronics or simply on a shelf near the equipment.

## CODED ASCII MATRIX DATA

Only ASCII [printable] characters are transmitted to and from the remote LED display, making debugging easy when using a serial terminal. The keypad simply transmits ASCII O-9, A, B, or V whenever the appropriate key is pressed.

The received LED data is a bit more complicated. There are in fact 10 possible columns (O-9) of data which could be passed to the display. To keep things straight, I use a two-byte ASCII transmission for each half column. The first byte is the column number (O-9)+20h. The second byte is the nybble value (O-F)+40h (for the lower nybble of data), or (O-F)+50h (for the upper nybble of data). This might seem a bit complex, but notice each byte is a

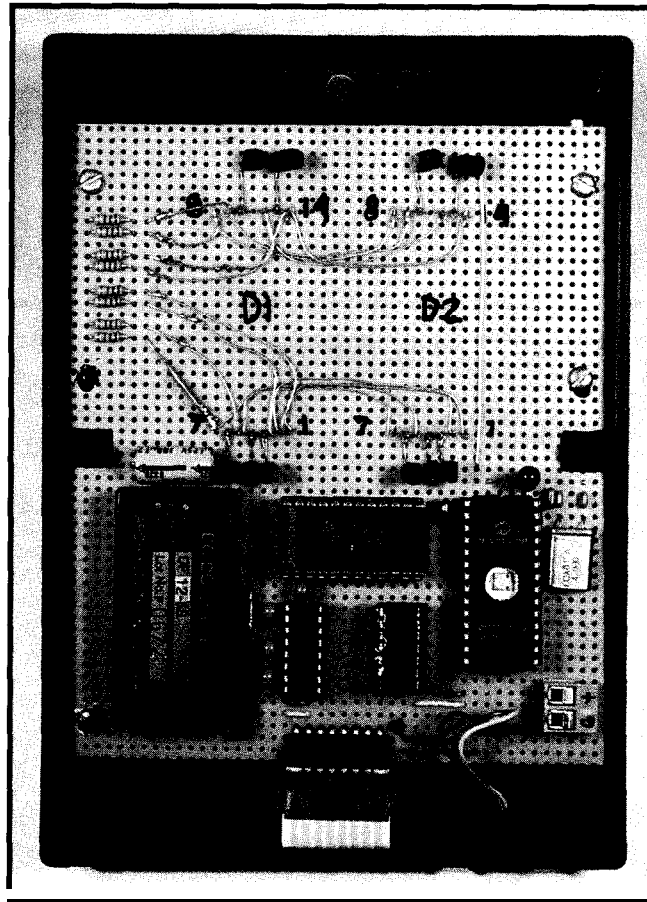


Photo 2—The back of the remote shows the PIC (right), multiplexer (center), and DC-DC converter (left) that efficiently converts the 12 volts supplied to it to the necessary 5 volts. The schematic in Figure 2 shows the use of an LM7805, which is a cheaper, smaller, and hotter solution.

printable character and you can tell at a glance what the data is and where it goes.

The display can actually be updated very rapidly and may bring to mind other uses for this technique from information displays to electronic art.

## A PINCH OF HARDWARE

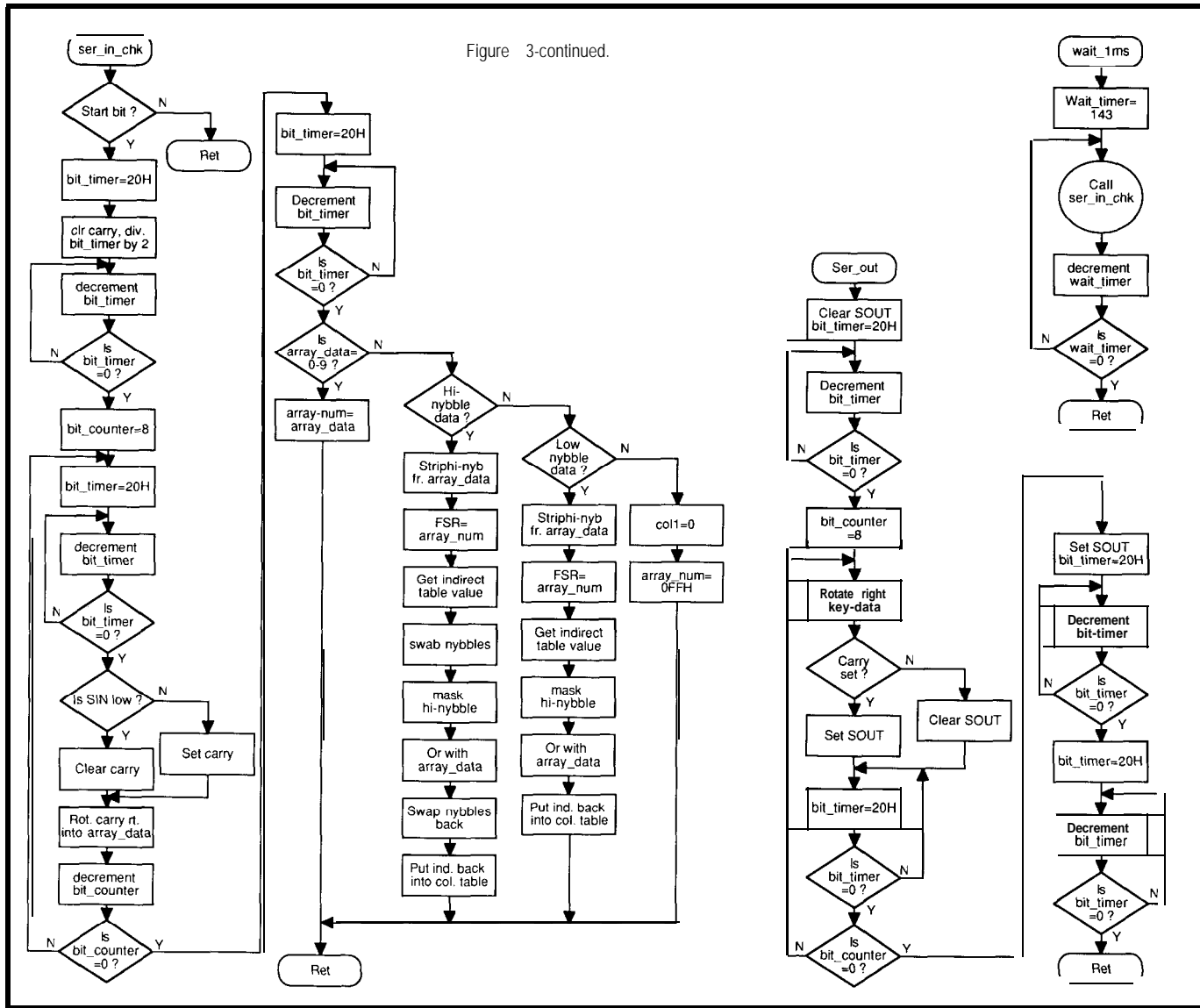
Little is needed beyond a micro, a MAX232, and a 4-to-16-line decoder to handle serial data, keypad scanning, and LED matrix multiplexing (see Figure 2). The 10 columns of LEDs and 3 columns of the keypad are enabled one at a time by the 74HCT154. The LED's "row data" is output on Port B of the micro. The lower half of Port C drives the '154 to select which column is enabled. The upper half of Port C uses the keypad rows as inputs (they are held high with pull-ups.) Port A is used for communication (RX and TX). See Figure 3.

The upper three enables [of the '154] are not needed, so its addressing begins with address 12 (decimal),

which is the first keypad column. If a key in the first column is being pressed, the associated row will be pulled to ground. The code looks for a low on any keypad input during the time in which the column is enabled. If a key press has been detected during any of the three column enables and no key was down during the last check, then we jump to the serial output routine and transmit one character at 9600 bps. If no keys were pressed, then the code moves on to scan the LED column drivers. Row output data is updated as each of the columns are enabled. A short pause of 1 millisecond is executed between each column enable to allow enough time to pass so as to let the illuminated LED stimulate the eye's retina enough to provide for persistence of vision. Once all the columns have been scanned, we go back and look for a key press once again. This cycle time also acts as a debouncing to the switches.

One more thing. We have to pause every now and then [actually quite

Figure 3-continued.



often] to look for a start bit coming into the micro. No interrupts are available, so polling must take place in a timely manner. By checking the RX pin after every column enable (and also during the 1 millisecond pause) we are assured of capturing the serial data within the allotted time frame. After the receipt of a character, a determination is made as to what to do with it. It can point to the appropriate column table offset, or if it's data, it can be placed into either the upper or lower nybble in the table. If the character falls outside the legal boundaries, it is tossed out and an error is displayed.

Errors are displayed by lighting the entire column of LEDs between the first column of command prompting LEDs and the last columns which make up the 8x8 matrix. New data

sent to the remote should always turn off the error LEDs to clear any error status.

### A LAST LOOK

Did we actually succeed in creating a piece of equipment which satisfied both of our needs? Well, Steve has already completed the integration of the AVmux into his new entertainment room and although I can't smell the popcorn from here, I have seen "Top Gun" vapor trails originating from high atop "Ciarcia Peak." As far as my installation goes, I am looking for a few more video sources to add. I just can't bear to leave those extra video inputs unconnected. 📺

Jeff Bachiochi (pronounced "BAH-key-AH-key") is an electrical engineer on

the Computer Applications Journal's engineering staff, His background includes product design and manufacturing. He can be reached at [jeff.bachiochi@circellar.com](mailto:jeff.bachiochi@circellar.com).

### SOFTWARE

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

### IRS

- 416 Very Useful
- 417 Moderately Useful
- 418 Not Useful