BUILD A SERIAL EPROM PROGRAMMER

BY STEVE CIARCIA

An inexpensive way to put your programs on a chip

Over the years, many articles have been published on programming EPROMs (erasable programmable read-only memories). The number of articles alone indicates the value of an EPROM programmer and the interest expressed in the subject. True-blooded computer experimenters consider an EPROM programmer as essential a tool as a soldering iron and a DVM (digital voltmeter).

Most EPROM programmers designed for personal computers are implemented as bus-dependent I/O (input/output) peripheral cards that use computer-specific, machine-language driver programs. By eliminating the need for an enclosure and using the system power supply, a relatively cost-effective unit can be produced. Unfortunately, if I designed such a unit, it probably wouldn't be for the computer you own.

For computer users who don't have expansion buses or who want their EPROM programmer to be transportable between systems, the only alternative is a stand-alone EPROM programmer attached to a serial port (much like a modem). Making it a separate peripheral device, however, necessarily increases its cost. In fact, external serial-port EPROM programmers are frequently two or three times the cost of board-level units.

A certain portion of the cost is due to its separate power supply and enclosure, but most of the expense is attributed to the features that manufacturers generally incorporate in the devices. The majority of stand-alone serial-connected programmers are, in fact, designed as intelligent EPROM programmers that have the basic processing power and memory of whole computers. I have taken this approach on previous designs. Such devices perform well and require little assistance from the host system beyond the data to be programmed.

This time I'm approaching the problem differently. I've decided to keep it simple and design the most universally applicable and cost-effective programmer that I can.

The latest Circuit Cellar EPROM programmer is a serial-port programmer that has the speed of a turtle, the intelligence of the mightiest computer (that is, it has absolutely no smarts of its own), and is as functional as a doorstop between uses. On the positive side, it's fully documented, universally applicable, and easily expandable to ac-

(continued)

Steve Ciarcia (pronounced "see-ARE-see-ah") is an electronics engineer and computer consultant with experience in process control, digital design, nuclear instrumentation, and product development. He is the author of several books about electronics. You can write to him at POB 582, *Glastonbury,* CT 06033.

commodate future EPROM types.

The serial-port programmer can be operated from almost any system with a serial port. The driver software is written completely in BASIC with no machine-language routines. The serial-port programmer offers all the hardware features to program 2716, 2732, 2732A, 2764, and 27128 EPROMS through a serial port, including: RS-232C compatibility, no handshaking necessary, internal power supplies, jumper-selectable EPROM types, and jumper-selectable data rates.

The BASIC-language driver program included offers features such as:

- menu-driven operation using single keystrokes
- a help routine that can be called at any time
- single-byte or burst-write modes
- read or copy EPROM
- optional programming from a disk file
- verify after write
- verify EPROM erasure
- screen-dump routines by page or byte
- single-stepping mode
- software-controlled read/write mode select
- BASIC driver that can be user-modified

Reviewing EPROM Basics

A personal computer, even in its minimum configuration, always contains some user-programmable memory or RAM (random-access read/write memory), usually in the form of semiconductor-memory integrated circuits. This memory can contain both programs and data and can be read or modified as needed.

Any of several kinds of electronic components can function as bit-storage elements in this kind of memory. TTL (transistor-transistor logic) type-7474 flip-flops, bistable relays, or tiny ferrite toroids (memory cores) are suitable, but they all cost too much, are hard to use, and have other disadvantages.

In personal computer and other microprocessor-based applications, the most cost-effective memory is made from MOS (metal-oxide semiconductor) ICs (integrated circuits). Unfortunately, data stored in these semiconductor RAMs is volatile. When the power is turned off, the data is lost. Many ways of dealing with this problem have been devised, with essential programs and data usually stored in some nonvolatile medium.

In most computer systems, some data or programs are stored in nonvolatile ROM (read-only memory). A semiconductor ROM can be randomly accessed for reading in the same manner as the volatile memory, but the data in the ROM is permanent. In a mask-programmed ROM, the data that can be read is determined during the manufacturing process. Whenever power is supplied to the ROM, this permanent data (or program) is available. In small computer systems, ROM is chiefly used to contain operating systems and/or BASIC interpreters—programs that don't need to be changed.

Another type of ROM is the PROM (programmable read-only memory). A PROM component is delivered containing no data. The user decides what data it should contain and permanently programs it with a special programming device. Once initially programmed, PROMs exhibit the characteristics of mask-programmed ROMs. You might label such PROMs as write-once memories.

The EPROM, which is ultraviolet-light-erasable, is a compromise between the write-once kind of PROM and the volatile memory. You can think of the EPROM as a read-mostly memory, used in read-only mode most of the time but occasionally erased and reprogrammed as necessary. The EPROM is erased by exposing the silicon chip to ultraviolet light at a wavelength of 2537 angstroms. Conveniently, most EPROM chips are packaged in an enclosure with a transparent quartz window.

How an EPROM Works

EPROMs store data bits in cells formed from stored-charge FAMOS (floating-gate avalanche-injection metal-oxide semiconductor) transistors. Such transistors are similar to positive-channel silicon-gate field-effect transistors, but they have two gates. The lower or *floating* gate is completely surrounded by an insulator layer of silicon dioxide; the upper *control* or *select* gate is connected to external circuitry.

The amount of electric charge stored on the floating gate determines whether the bit cell contains a 1 or a 0. Charged cells are read as 0s; uncharged cells are read as 1s. When the EPROM chip comes from the factory, all bit locations are cleared of charge and are read as logic 1s; each byte contains hexadecimal FF.

When a given bit cell is to be burned from a 1 to a 0, a current is passed through the transistor's channel from the source to the gate. (The electrons, of course, move the opposite way.) At the same time, a relatively high voltage potential is placed on the transistor's upper select gate, creating a strong electric field within the layers of semiconductor

(continued)

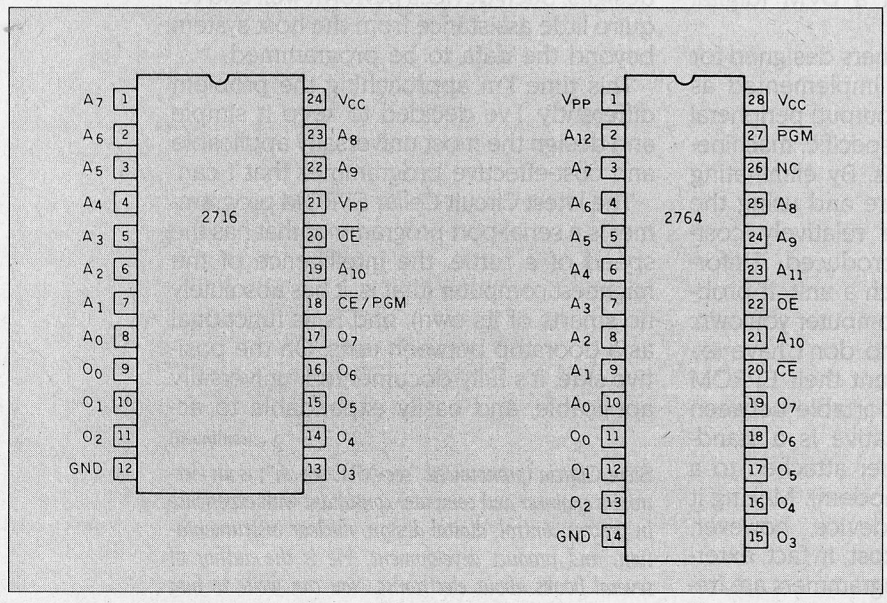
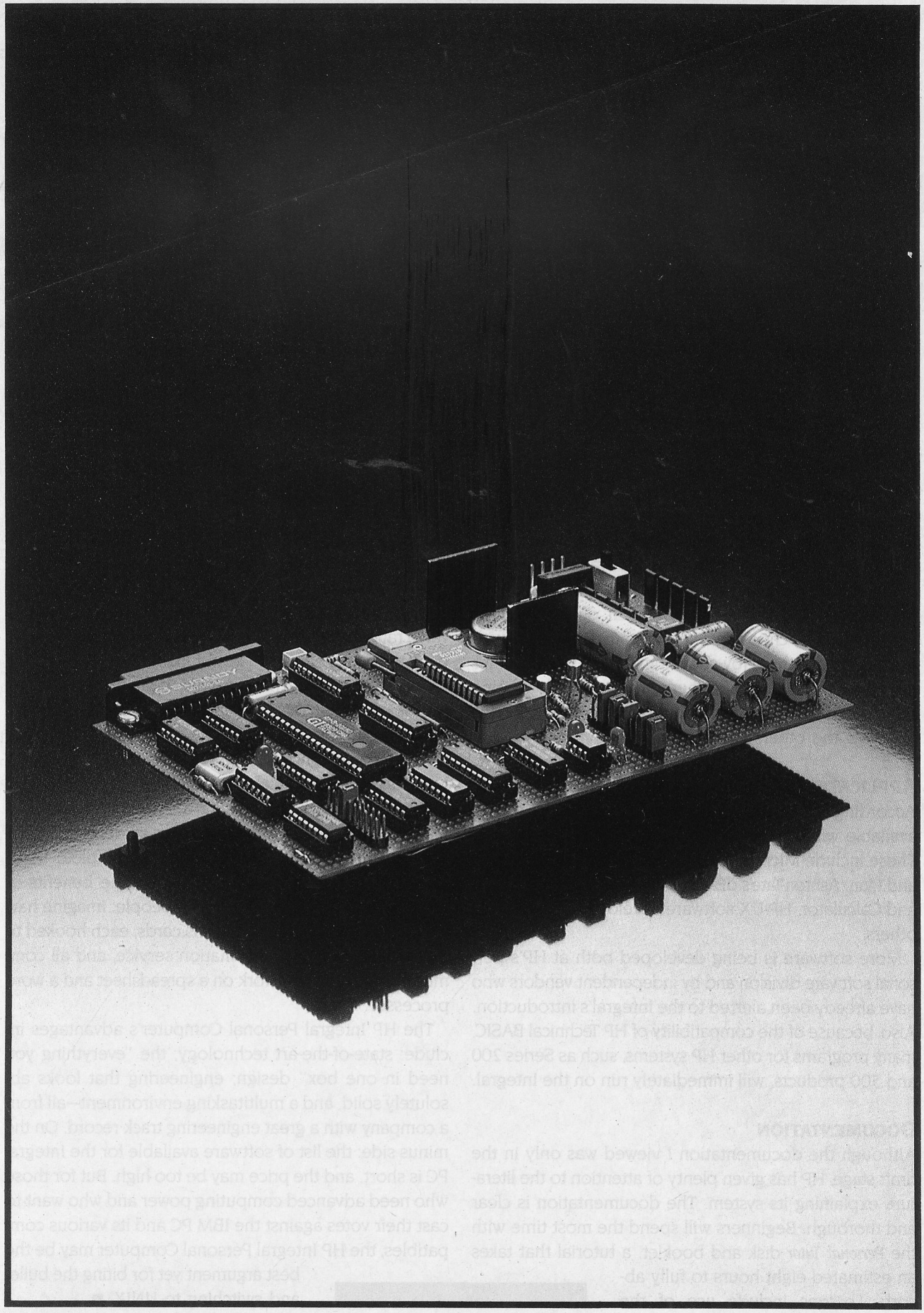


Figure 1: *Pinouts of the* 2716 *and* 2764 EPROMs.



PHOTOGRAPHED BY PAUL AVIS.

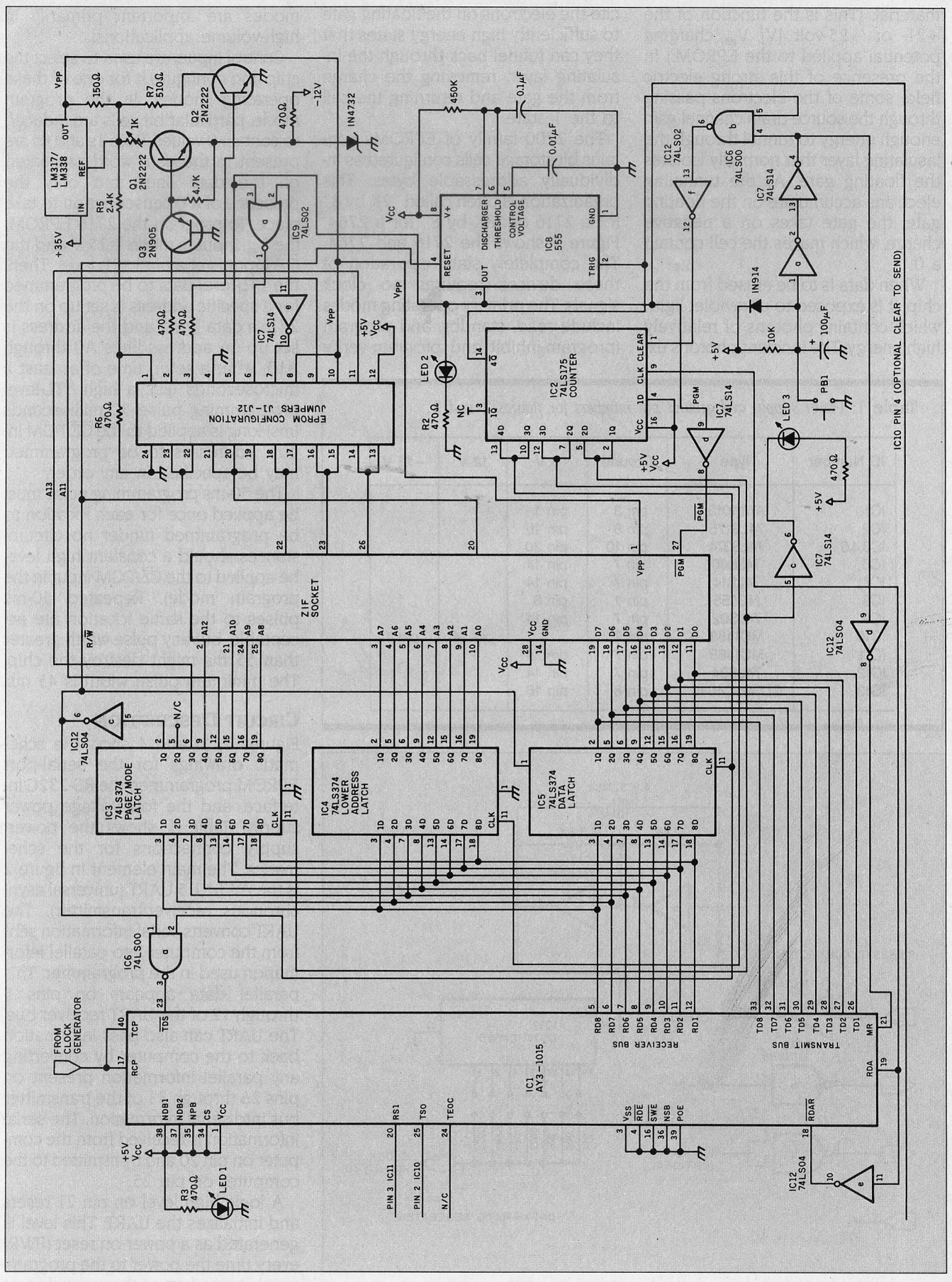


Figure 2: *The serial-port* EPROM *programmer.*

material. (This is the function of the +21- or +25-volt [V] V_{pp} charging potential applied to the EPROM.) In the presence of this strong electric field, some of the electrons passing through the source-drain channel gain enough energy to tunnel through the insulating layer that normally isolates the floating gate. As the tunneling electrons accumulate on the floating gate, the gate takes on a negative charge, which makes the cell contain a 0.

When data is to be erased from the chip, it is exposed to ultraviolet light, which contains photons of relatively high energy. The incident photons excite the electrons on the floating gate to sufficiently high energy states that they can tunnel back through the insulating layer, removing the charge from the gate and returning the cell to the 1 state.

The 2700 family of EPROMs contains bit-storage cells configured as individually addressable bytes. This organization is often called "2K by 8" for a 2716 or "8K by 8" for a 2764. Figure 1 shows the 2716 and 2764. The completely static operation of these devices requires no clock signals. The primary operating modes include read, standby, and program (program-inhibit and program-verify modes are important primarily in high-volume applications).

Control inputs are used to select the chip and configure it for one of these operating modes. In the program mode, particular bit cells are induced to contain 0 values. Both 1s and 0s are present in the data word presented on the data lines, but only the presence of a 0 causes action to take place. To program the 2716 EPROM, the V_{pp} input is made +25 V and the $\overline{OE}$ input is at a high TTL level. Then, the TTL-level data to be programmed for a specific address is set up on the 2716's data lines, and the address is set up on address lines A0 through A10. After a setup time of at least 2 microseconds (μs), a high TTL-level programming pulse 50 milliseconds (ms) long is applied to the $\overline{CE}$/PGM input. Addresses to be programmed may be specified in any order.

The 50-ms programming pulse must be applied once for each location to be programmed (under no circumstances should a constant high level be applied to the $\overline{CE}$/PGM input in the program mode). Repeated 50-ms pulses to the same location are acceptable, but any pulse width greater than 55 ms might destroy the chip. The minimum pulse width is 45 ms.

Table 1: *Power supply and ground pin numbers for figures 2 and 3.*

IC Number	Type	Ground	5 V	12 V	−12 V
IC1	AY3-1015	pin 3	pin 1		
IC2	74LS175	pin 8	pin 16		
IC3,4,5	74LS374	pin 10	pin 20		
IC6	74LS00	pin 7	pin 14		
IC7	74LS14	pin 7	pin 14		
IC8	NE555	pin 1	pin 8		
IC9	74LS02	pin 7	pin 14		
IC10	MC1488	pin 7		pin 14	pin 1
IC11	MC1489	pin 7	pin 14		
IC12	74LS04	pin 7	pin 14		
IC13	CD74HC4040	pin 8	pin 16		

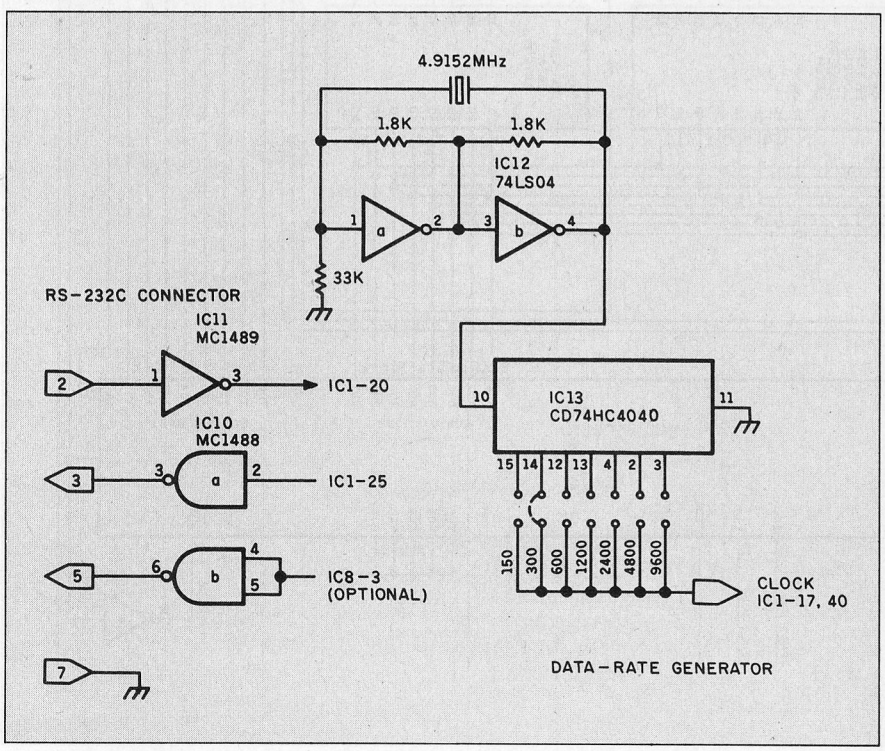


Figure 3: *Serial interface and data-rate generator.*

Circuit Description

Figures 2, 3, and 4 show the schematic drawings for the serial-port EPROM programmer, the RS-232C interface, and the four-voltage power supply. Table 1 shows the power-supply connections for the schematics. The main element in figure 2 is the AY-3-1015 UART (universal asynchronous receiver/transmitter). The UART converts serial information sent from the computer into parallel information used in the programmer. This parallel data appears on pins 5 through 12 of the UART receiver bus. The UART can also pass information back to the computer by converting any parallel information present on pins 26 through 33 of the transmitter bus into serial information. The serial information is received from the computer on pin 20 and transmitted to the computer on pin 25.

A logic high level on pin 21 resets and initializes the UART. This level is generated as a power-on reset (PWR) every time the power to the programmer is turned on or the manual reset button pressed. This PWR also clears

the receiver character counter, IC2.

UART pins 35 through 39 set the format of the serial transmission between the computer and the programmer. (I chose to hard-wire these options rather than provide option switches that are rarely used.) As shown, the UART is configured for an 8-bit character length with 1 stop bit and parity checking inhibited. If your computer requires 2 stop bits, connect pin 36 to +5 V instead of ground. The programmer will operate at any desired data rate up to and including 9600 bits per second (bps). A software delay loop keeps the programmer from being swamped.

The programmer requires 4 bytes to be sent from the computer for each location read from or written to in the EPROM. This 4-byte protocol eliminates the need for incremental counters and sophisticated decision logic in the programmer. It does, however, reduce the speed of read and erasure-verification operations.

The first 3 bytes received are latched a byte at a time into latches IC3, IC4, and IC5. The latching pulses are generated by IC2, which is configured as a 4-bit byte counter. Each time a byte is received by the UART, an RDA (received data available) pulse is generated at pin 19 of the UART. This pulse is used to clock IC2 and is gated back to the RDAV (reset data available) line, pin 18, to clear the receiver section of the UART. As the counter clocks, the leading edges of its output latch the data from the UART into IC3, IC4, or IC5. The counter is reset by the PWR line or when the fourth byte is received.

The first byte received by the programmer contains the most significant 3 to 6 bits of the EPROM address (depending upon the EPROM type) and 1 bit to select either the read or write mode of operation. A logic 1 in bit 7 sets the write mode; a logic 0 sets the read mode.

The second byte contains the lower 8 bits of the EPROM address.

The third byte contains the data to be programmed into the addressed location when it is in the write mode or a dummy character when in the read mode.

The fourth byte contains dummy data in both the read and write modes. When the counter increments with the reception of the fourth byte, it causes IC2 to reset. The time between setting this output bit and clearing the counter is about 100 nanoseconds (ns). This short pulse concluding the setup of the address and data is used to trigger the actual programming pulse to the EPROM.

The programming pulse to the EPROM is generated by IC8, which is configured as a 50-ms one-shot (triggered by the reception of the fourth byte). The programming pulse is fed to the EPROM at several different locations, depending on which EPROM is being programmed and how the EPROM selection jumper block (see figure 5) is configured.

The one-shot is functional only when the mode select line (R/$\overline{W}$, read/not write) IC3 pin 2 is a logic 0, setting the write mode. The mode select line is also used to select the programming voltage ranges of the various EPROMs. When configured for a 2732 or a 2716 EPROM, a low on the mode select line sets the V_{pp} supply to a 25-V level. For all other EPROM types, the V_{pp} supply is set to a 21-V level.

Depending on the configuration of the jumper block, the mode select line sets the proper TTL levels at the $\overline{CE}$ and $\overline{OE}$ pins to place the various EPROMs in the read or write mode. A logic high on the mode select line causes the V_{pp} supply to drop to 0 V for the 2732 and 2732A EPROMs and to 5 V for the other types.

The mode select line also functions as the output enable line of data latch IC5. When the programmer is in the write mode, data from the UART is latched and directed to the EPROM data bus for programming. When the programmer is in the read mode, IC5's output is disabled, and the EPROM data-bus contents are transmitted back to the computer.

LEDs (light-emitting diodes) 1, 2, and 3 indicate when power is on and when read and write pulses occur. They are not necessary to the operation of the programmer and are merely included as visual aids.

Figure 3 shows the serial-interface connections and the data-rate generator. IC10 and IC11 are standard RS-232C transmitter and receiver chips that conform to the EIA (Electronic Industries Association) standard for RS-232C transmission. (If your computer needs a handshaking signal, the 50-ms write pulse can be connected to the clear-to-send line. It is not used with the software presented in this article.) The serial-communication rate between the programmer and the computer is jumper-selectable. A 4.9152-MHz oscillator is divided down through a CD74HC4040 (it must include the HC designation to accommodate the high frequency) to produce the appropriate clock rate for the UART.

Figure 4 shows the power supply used with the programmer. The power transformer I chose was 22 V CT (center tap), but any transformer from

(continued)

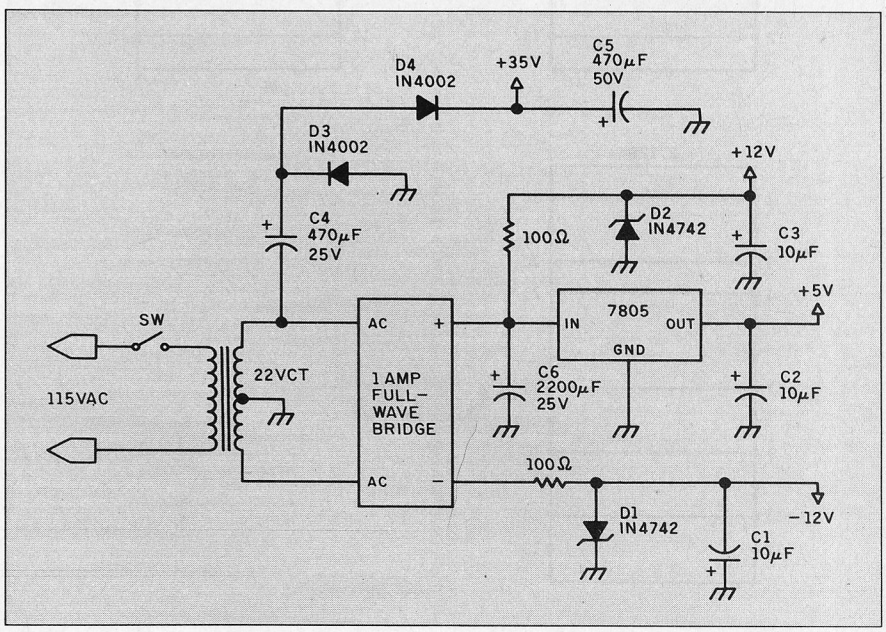


Figure 4: *Power supply.*

22 to 25.6 V CT is adequate. The secondary output of the transformer is full-wave rectified, filtered, and then regulated to +12 V, +5 V, and −12 V. Only the +5-V supply needs an actual IC regulator; less stringent zener regulation is adequate for the 12-V supplies to the RS-232C drivers.

The 35-V output consists of components C4, C5, D3, and D4 connected as a cascade voltage doubler with half-wave rectification. This configuration produces an input of approximately 32 to 34 V to the LM317/338 regulator. The minimum acceptable voltage at the input is 28.5 V (for a 25-V output). If you use a higher-output transformer than 22 V CT, be careful that the input to the V_{pp} regulator doesn't exceed 35 V. If it does, additional preregulation may be necessary to use this circuit.

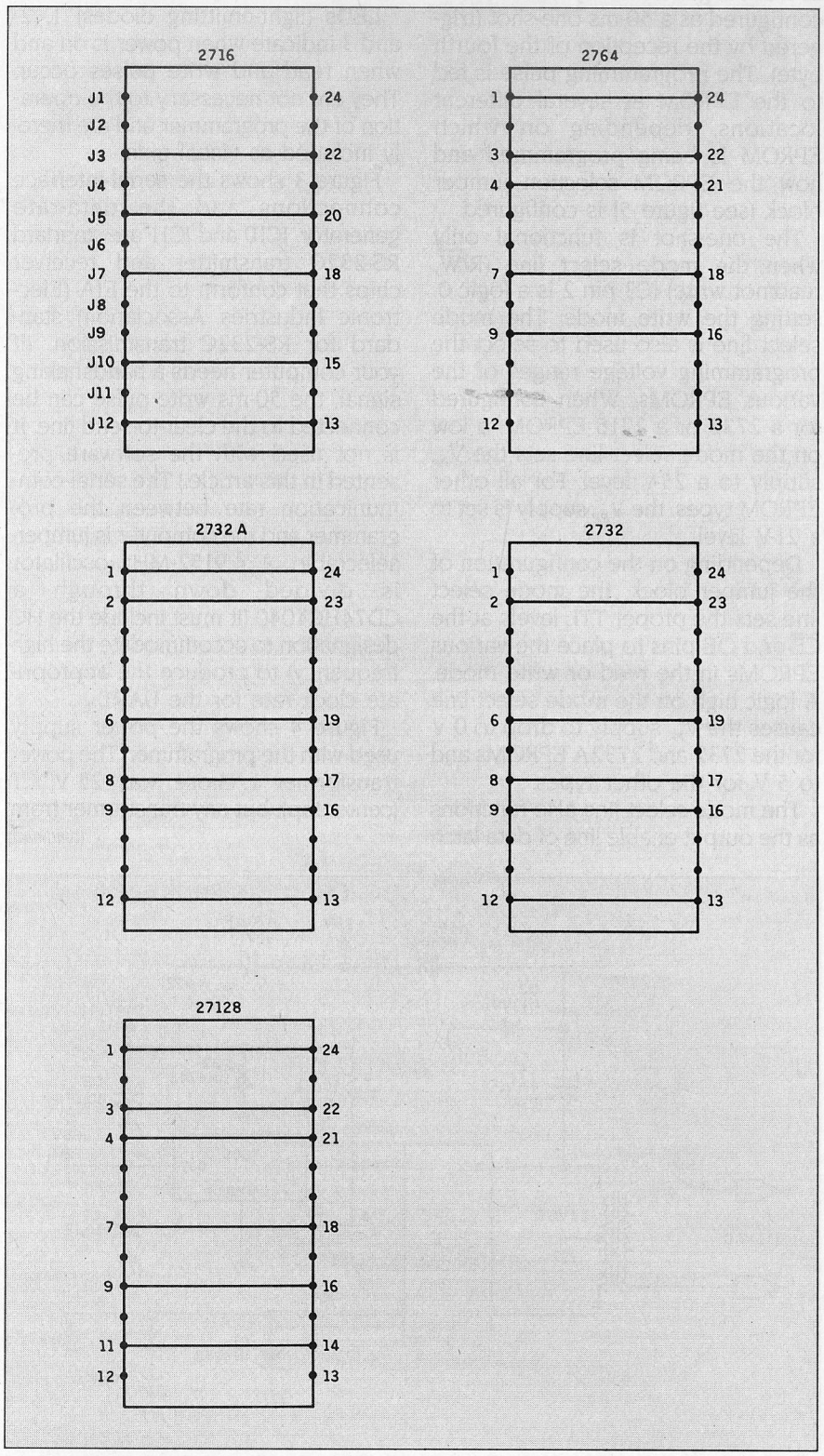


Figure 5: *Configuration jumpers.*

Figure 6 shows the programmable V_{pp} supply. The 2732A EPROM requires the programming voltage to be pulsed between 0 and 21 V, while a 2716 requires a pulse between 5 and 25 V. The supply is controlled by the jumper connections and the mode select line. With jumper #1 across R6, the supply is configured for a maximum V_{pp} level of 21 V. When it is removed, the supply has a maximum voltage of 25 V.

The minimum V_{pp} level is set by two jumper-selectable programming circuits, which are also connected to the regulator's output set point-adjust line. When jumper #2 is installed, a two-transistor circuit is enabled, which applies −1.2 V to the adjust line. The result is a 0-V output from the regulator. When jumper #3 is installed, the reference-adjust line is set to allow a +5-V regulator output.

Interacting with Hardware

The operation of the serial programmer should become clear by following an example of a write operation followed by a read operation. This is the sequence that would necessarily occur during a standard write-and-verify cycle.

First, the EPROM programmer is cleared and set to the read mode by the power-on reset pulse (which can be generated by pressing a button or by turning the programmer on) so that it is ready to receive the first character. If we plan a write cycle, the first character must contain a logic 1 in bit 8 to activate the write mode. The upper 3 to 6 bits of the EPROM address (the page address that depends on the size of the EPROM) must also appear in the first 3 to 6 bits (bit 0 through bit 5) of this first character. Each character of data to be programmed into the EPROM is sent to the programmer as a 4-byte transmission with the programming address specified each time.

Table 2 indicates the allowable bit patterns for this first character received by the programmer.

For our example, assume that the

data byte C3 (hexadecimal) is to be written into the first byte of page 4 in the EPROM. In this case, the first character received by the programmer should be 1x000100. The receipt of this character pulses IC2 and latches the page address and mode select bit into the page/mode latch, IC3. The mode select bit selects the EPROM for a write cycle, turns on the V_{pp} supply to the EPROM, releases the reset line on the timer, activates the output enable line of the data latch, and shuts off the transmission gate of the UART.

The second character sent contains the lower 8 address bits for the EPROM. To program the first location in page 4, the rest of the address must then be 00000000. This character sets the second stage of the counter and latches the lower address location into the lower address latch, IC4.

The third character, 11000011 (C3 hexadecimal), contains data to be programmed into the EPROM. When this character is received, the counter latches the data into the data latch, IC5.

The fourth character sent is a dummy character that may contain any value. This fourth and last character simply clocks IC2 and triggers the 50-ms programming pulse. When the one-shot times out, the programmer is still in the write mode. It has to be set to the read mode by initiating a read cycle.

The four characters sent in our present example of a write sequence are 1x000100, which sets the write mode and upper address; 00000000, which sets the lower address; 11000011, which sets the data byte (C3 hexadecimal); and xxxxxxxx, dummy data.

The read sequence is similar to the write sequence. The first character

(continued)

Table 2: *Allowable bit patterns.*

Page	Write Mode	Read Mode
0	1x000000	0x000000
1	1x000001	0x000001
2	1x000010	0x000010
3	1x000011	0x000011
4	1x000100	0x000100
5	1x000101	0x000101
6	1x000110	0x000110
7	1x000111	0x000111
.	.	.
.	.	.
64	1x111111	0x111111

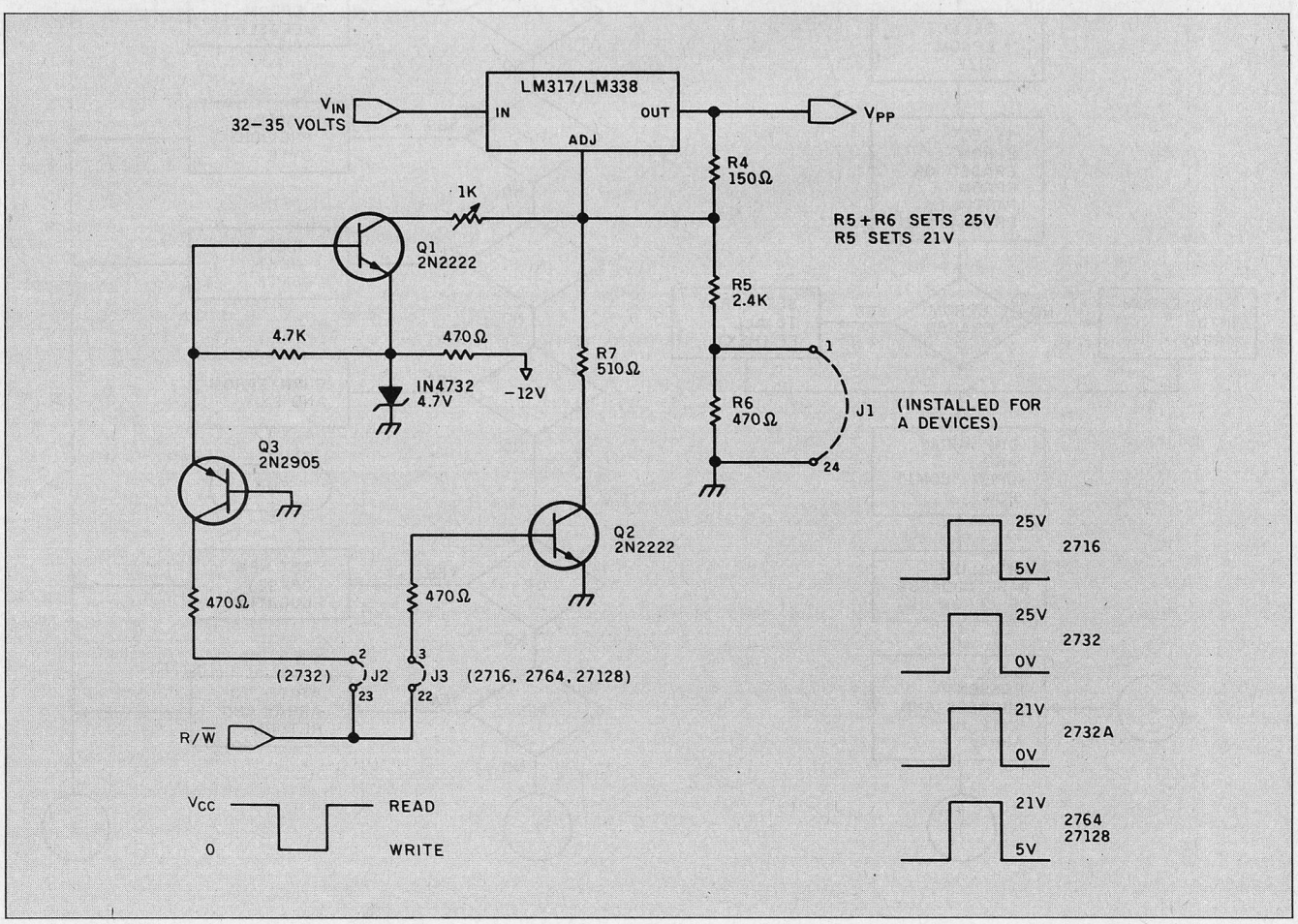


Figure 6: *Close-up of the programmable V_{pp} supply.*

sent again contains the upper bits of the address, but bit 8 is now set to logic 0 to put the mode select line high (read mode). A logic 1 on the read/write line deactivates the programming one-shot and tristates the data latch, IC5.

Again, the first character is latched into the page/mode latch, and the second character is latched into the lower address latch. With IC5 tristated, the EPROM's data output is placed on the UART transmitter bus. The third character is a dummy character that is used to clock IC2. This signal causes the UART to transmit the data on the transmitter bus to the computer. The fourth character is then sent to the programmer to reset the counter.

The four characters that must be sent in the verify sequence of our example are 0x000100, which sets the read mode and upper page address; 00000000, which sets the lower address; xxxxxxxx, which gets the data byte from the EPROM (C3 hexadecimal); and xxxxxxxx, which resets the programmer.

Programmer Software

The driver program shown in listing 1 could have been written in any language that supports input and output ports. [*This program is available for downloading from* BYTE*net Listings at* (603) 924-9820. *You can also receive it by sending an* IBM PC*-formatted disk and return postage to Steve Ciarcia.*] BASIC was chosen because it has wide appeal in the personal computer field and because most systems with serial I/O ports support BASIC. The software (flow-diagramed in figure 7) was written specifically for the IBM PC but can be easily modified to conform to most other systems that also support Microsoft BASIC. The program was written with a short MAIN program module that calls a number of subroutine modules. This modular approach makes modifying, debugging, or ex-

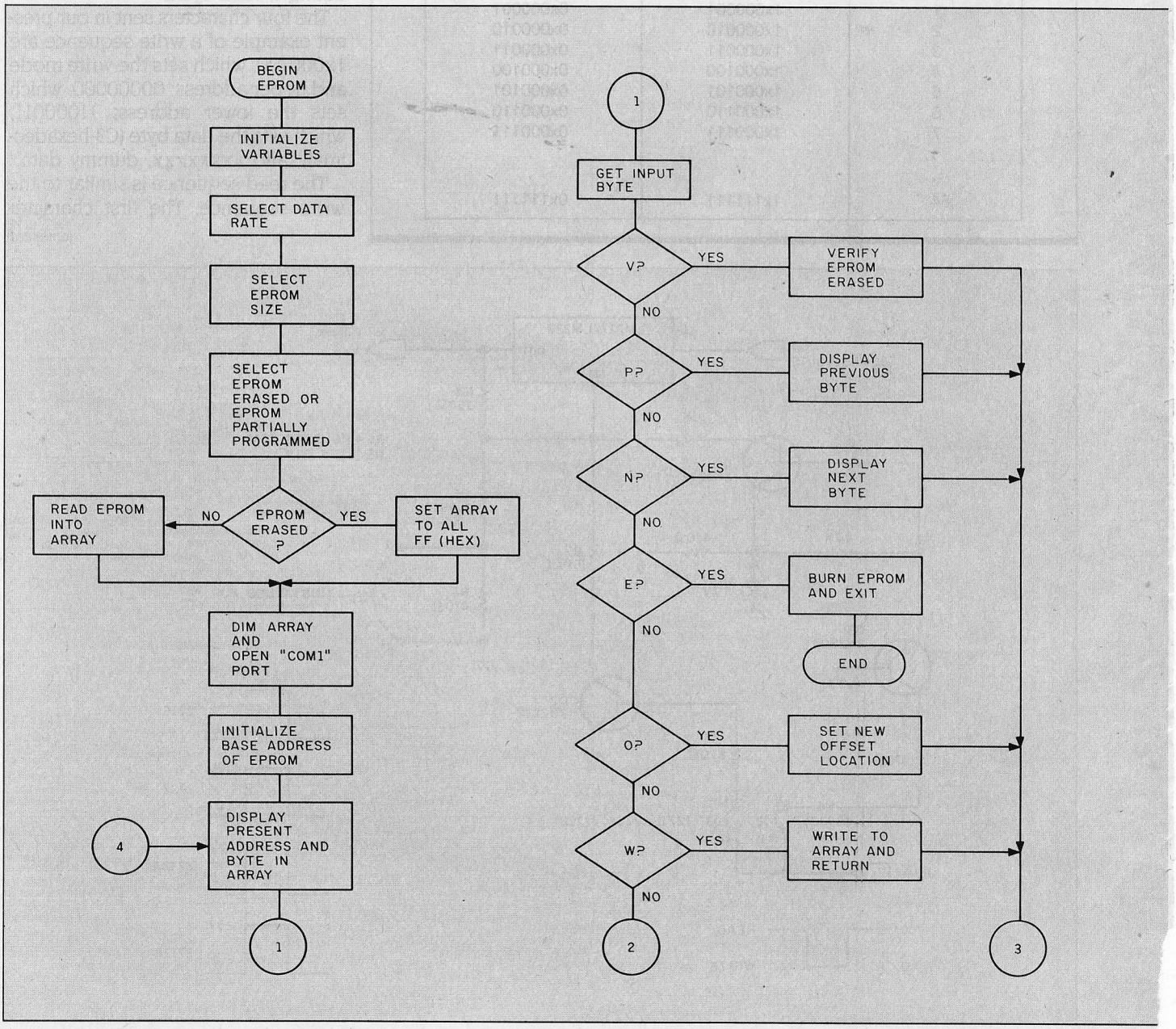


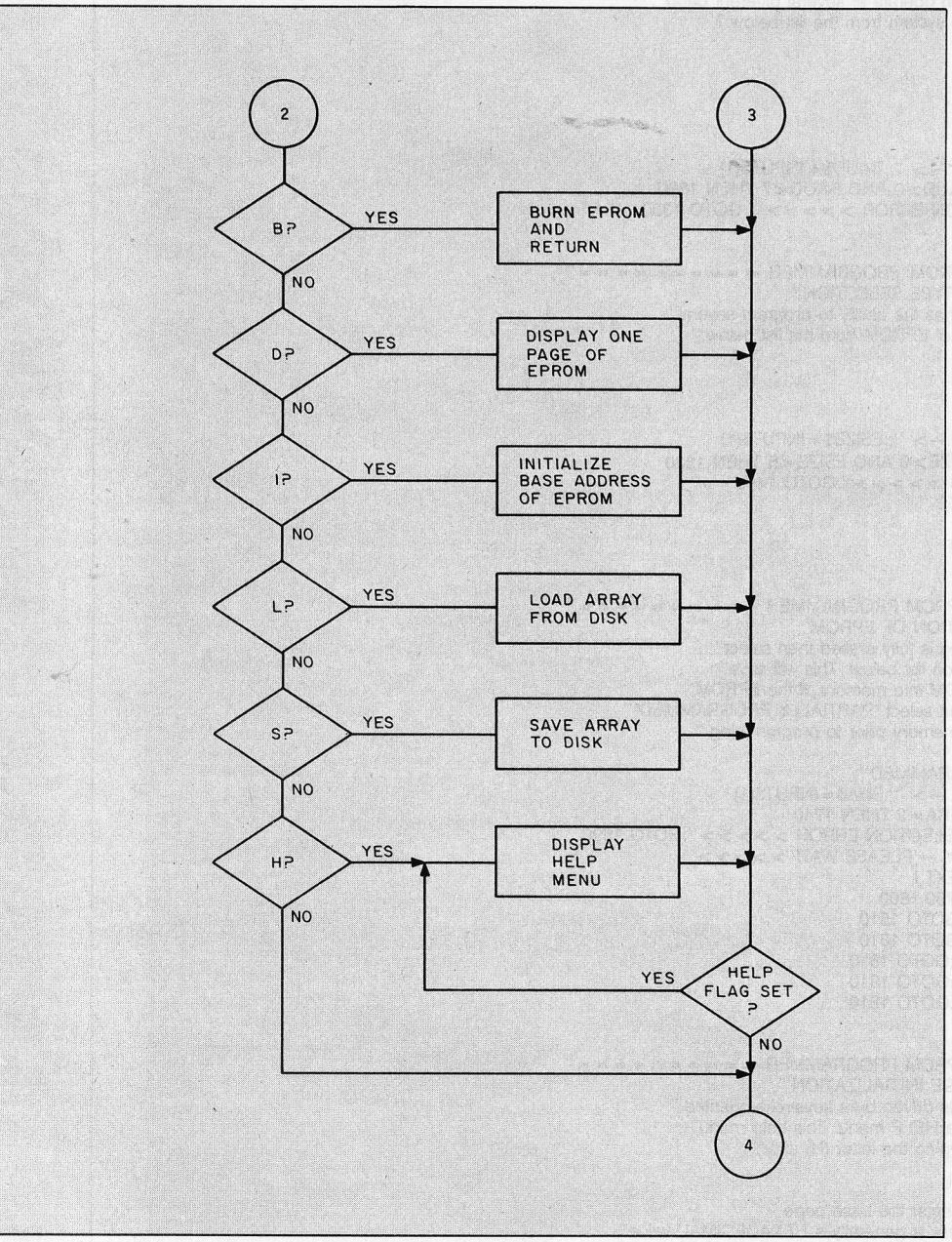
Figure 7: A *flowchart of the driver program.*

panding the software a much easier task. Examining the driver software should provide enough understanding so that any additions or changes desired can be easily implemented.

The program modules that access the serial port are labeled READ A BYTE and WRITE A BYTE in listing 1. These sections contain the only software modules that are hardware-dependent and that need to be configured to your particular system.

The WRITE module performs the actual program burn of the data into the EPROM. The first statement sends the page address to the serial port with the value of bit 8 set to 1. This is accomplished by combining the page address with the value 128 (10000000 binary). The page address is calculated elsewhere in the program before entering this module. The next statement sends the lower address contained in the variable BYTE to the serial port. This value is also calculated by the program prior to entering the WRITE module.

The statement "PRINT #3,DATUM" sends the data to be written into the EPROM to the serial port. The last statement in the WRITE module is a timing loop that causes the program to pause while the 50-ms timer in the serial-port programmer times out.



The READ module requests a data byte from the programmer and receives the byte from the serial port. It accomplishes this by sending a page address and byte address to the serial port as in the WRITE module. In this case, bit 8 of the page address is set to 0 to inform the programmer that a read cycle is being performed. The next two lines send a dummy data value and a strobe to the serial port to complete the read sequence. The values of DUMMY and STROBE are set in the INITIALIZATION module. The data sent by the serial-port programmer is received in the variable RDATA.

Once these modules have been configured to your system, it is a simple matter to write and read data from the programmer. Simply define the PAGE and BYTE address variables along with the DATUM value and send them to your serial port by calling the appropriate module. The rest of the program in listing 1 shows methods for doing this.

The approach used in the program is to place any data to be programmed into the EPROM in an array so that it can be reviewed and edited prior to burning it permanently into the EPROM. The array name is appropriately called ARRAY( ). The high-order byte of every element in ARRAY() stores a flag bit indicating that the lower-order byte of the element is data to be programmed. This method allows the program to write to only those locations in the EPROM where a valid data value has been entered in ARRAY().

Each time a data value is put into ARRAY(), the value is combined with 256 to set the flag. When it is time to send all the data to the EPROM, the flag is checked in each element, and only those elements with the flag bit set are sent to the EPROM. This process is repeated until all the flagged elements have been programmed. The initial values for ARRAY() are taken directly from the EPROM by reading each location and storing the values in ARRAY().

Several methods of entering data into ARRAY() are used in the program. One method is to enter each data value directly from the keyboard; another method is to fill ARRAY() by reading an already-programmed EPROM. Finally, a disk file previously

(continued)

Listing 1: EPROM *programmer routines.*

```
1000 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
1010 REM                    SERIAL EPROM PROGRAMMER
1020 REM                          written in
1030 REM                  MICROSOFT BASIC for the IBM PC
1060 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
1070 REM INITIALIZATION ROUTINE
1090 KEY OFF
1100 LINE25$ = "BAUD RATE = \  \EPROM = \  \ BASE PAGE = \  \"
1110 BR$ = "0000":EP$ = BR$:BP$ = BR$
1120 DEFINT A – Z:ON ERROR GOTO 4600
1130 STROBE = 255:DUMMY = 255:PAGE = 0:BYTE = 0:DATUM = 255
1140 K$ = "VPNEOWHDIBSL":FORMAT$ = "PAGE = \\  BYTE = \\  DATA = \\"
1150 MIMAGE = 0:MCRADDR = &H3FC:DELAY = 100
1160 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
1170 REM MAIN BODY OF PROGRAM — KEYBOARD SEQUENCE
1190 GOSUB 2250
1200 PRINT"= = = = = = = = = = SERIAL EPROM PROGRAMMER = = = = = = = = = = "
1210 PRINT"                          BAUD-RATE SELECTION"
1230 PRINT"The SERIAL PORT programmer can operate at several different baud"
1240 PRINT"rates. Select the baud rate for your system from the list below:"
1260 PRINT"       (1) 300 baud"
1270 PRINT"       (2) 600 baud"
1280 PRINT"       (3) 1200 baud"
1290 PRINT"       (4) 2400 baud"
1300 PRINT"       (5) 4800 baud"
1310 PRINT"       (6) 9600 baud"
1330 PRINT"Enter the number of your selection —> ";:BAUD$ = INPUT$(1)
1340 PRINT BAUD$:BAUD = VAL(BAUD$):IF BAUD>0 AND BAUD<7 THEN 1360
1350 PRINT"<<<<< BAUD-RATE SELECTION ERROR >>>>>": GOTO 1330
1360 BR$ = STR$(300*2^(BAUD – 1))
1370 GOSUB 2250
1380 PRINT"= = = = = = = = = = SERIAL EPROM PROGRAMMER = = = = = = = = = = "
1390 PRINT"                          EPROM-TYPE SELECTION"
1410 PRINT"The SERIAL EPROM programmer has the ability to program several"
1420 PRINT"different EPROMS. Select the type of EPROM from the list below:"
1440 PRINT"       (1) 2716"
1450 PRINT"       (2) 2732/2732A"
1460 PRINT"       (3) 2764"
1470 PRINT"       (4) 27128"
1490 PRINT"Enter the number of your selection —> ";:ESIZE$ = INPUT$(1)
1500 PRINT ESIZE$:ESIZE = VAL(ESIZE$):IF ESIZE>0 AND ESIZE<5 THEN 1520
1510 PRINT "<<<<< EPROM-TYPE ERROR >>>>>":GOTO 1490
1520 DSIZE = 1024*2^ESIZE:PAGES = DSIZE/256
1530 EP1$ = STR$(16*2^(ESIZE – 1))
1540 EP$ = "27" + RIGHT$(EP1$,LEN(EP1$) – 1)
1550 DIM ARRAY(DSIZE)
1560 GOSUB 2250:GOSUB 4790:GOSUB 2250
1570 PRINT "= = = = = = = = = = SERIAL EPROM PROGRAMMER = = = = = = = = = = "
1580 PRINT "                          CONDITION OF EPROM"
1600 PRINT"If the EPROM you are programming is fully erased then select"
1610 PRINT"'EPROM ERASED' from the selection list below. This will save"
1620 PRINT"the time required to read the EPROM into memory. If the EPROM"
1630 PRINT"has been partially programmed then select 'PARTIALLY PROGRAMMED'"
1640 PRINT"and the EPROM will be read into memory prior to programming."
1660 PRINT"     (1) EPROM ERASED"
1670 PRINT"     (2) EPROM PARTIALLY PROGRAMMED"
1690 PRINT"Enter the number of your selection —> ";:ERA$ = INPUT$(1)
1700 PRINT ERA$:PRINT:ERA = VAL(ERA$):IF ERA = 2 THEN 1740
1710 IF ERA<> 1 THEN PRINT"<<<<< SELECTION ERROR >>>>>":GOTO 1690
1720 PRINT"<<<<< INITIALIZING MEMORY — PLEASE WAIT >>>>>"
1730 FOR I = 0 TO DSIZE – 1:ARRAY(I) = 255:NEXT I
1740 ON BAUD GOTO 1750,1760,1770,1780,1790,1800
1750 OPEN "COM1:300,n,8,1,rs,cs,ds" AS #3:GOTO 1810
1760 OPEN "COM1:600,n,8,1,rs,cs,ds" AS #3:GOTO 1810
1770 OPEN "COM1:1200,n,8,1,rs,cs,ds" AS #3:GOTO 1810
1780 OPEN "COM1:2400,n,8,1,rs,cs,ds" AS #3:GOTO 1810
1790 OPEN "COM1:4800,n,8,1,rs,cs,ds" AS #3:GOTO 1810
1800 OPEN "COM1:9600,n,8,1,rs,cs,ds" AS #3
1810 GOSUB 2250
1820 PRINT "= = = = = = = = = = SERIAL EPROM PROGRAMMER = = = = = = = = = = "
1830 PRINT "                          BASE-PAGE INITIALIZATION"
1850 PRINT"The SERIAL EPROM programmer is driven by a keystroke-oriented"
1860 PRINT"program. The keys are defined in a HELP menu. This help menu"
1870 PRINT"can be displayed at any time by typing the letter (H) after"
1880 PRINT"the program has been initialized."
1890 PRINT:PRINT
1900 PRINT"To initialize the program you must enter the base page"
1910 PRINT"address of the EPROM. This address is generally a HEXADECIMAL value"
1920 PRINT"corresponding to the beginning page of an even 2K-byte boundary."
```

```
1930 PRINT"For example 00,08,B0,B8,etc."
1950 GOSUB 3770:REM                                    SET BASE ADDRESS
1960 IF HFLAG=1 THEN HFLAG=0:GOTO 1950
1970 IF ERA=1 THEN 2000
1980 PRINT"A MEMORY IMAGE OF YOUR EPROM IS BEING MADE"
1990 GOSUB 3890:REM                                    MAKE MEMORY IMAGE
2000 GOSUB 2880:REM                                    DISPLAY HELP MENU
2010 PRINT:PRINT
2020 PRINT"YOUR PRESENT LOCATION IS:"
2030 GOSUB 2320:REM                                    READ AND DISPLAY DATA
2040 PRINT"COMMAND —>";
2050 IKEY$=INPUT$(1)
2060 IF IKEY$>="a" AND IKEY$<="z" THEN IKEY$=CHR$(ASC(IKEY$) AND 95)
2070 K=INSTR(K$,IKEY$):IF K=0 THEN PRINT "WHAT ?";:GOTO 2050
2080 HFLAG=0
2090 ON K GOSUB 3430,2380,2440,2160,2500,2660,2880,3550,3760,3980,4240,4400
2100 REM            V    P    N    E    O    W    H    D    I    B    S    L
2110 IF HFLAG=1 THEN GOSUB 2880
2120 IF HFLAG=1 OR IKEY$="H" THEN 2010 ELSE 2030
2130 REM  ==========================================
2140 REM BURN EPROM AND END OPTION
2160 GOSUB 3980
2170 IF IKEY$<>"N" THEN RETURN
2180 CLOSE:END
2190 REM  ==========================================
2200 REM MAIN BODY ENDS HERE — SUBROUTINE MODULES FOLLOW
2220 REM  ==========================================
2230 REM DISPLAY STATUS LINE
2250 CLS:LOCATE 25,1:PRINT USING LINE25$;BR$,EP$,BP$;
2260 PRINT "COMMANDS: ";K$
2270 LOCATE 3,1,1:RETURN
2280 REM  ==========================================
2300 REM DISPLAY LOCATION AND DATA
2320 RDATA=ARRAY(PAGE*256+BYTE) AND 255:REM          GET DATUM FROM ARRAY
2330 PRINT USING FORMAT$;HEX$(BIAS+PAGE),HEX$(BYTE),HEX$(RDATA)
2340 RETURN
2350 REM  ==========================================
2360 REM DECREMENT ADDRESS
2380 IF PAGE=0 AND BYTE=0 THEN RETURN ELSE BYTE=BYTE-1
2390 IF BYTE=-1 THEN PAGE=PAGE-1:BYTE=255
2400 RETURN
2410 REM  ==========================================
2420 REM INCREMENT ADDRESS
2440 IF PAGE=PAGES-1 AND BYTE=255 THEN RETURN ELSE BYTE=BYTE+1
2450 IF BYTE=256 THEN PAGE=PAGE+1:BYTE=0
2460 RETURN
2470 REM  ==========================================
2480 REM OFFSET TO NEW STARTING ADDRESS
2500 ADD$=" ":PRINT:PRINT"ENTER NEW LOCATION IN HEXADECIMAL (hhhh) —> ";
2510 L$=INPUT$(1):PRINT L$;
2520 IF L$>="a" AND L$<="z" THEN L$=CHR$(ASC(L$) AND 95)
2530 IF L$="H" THEN HFLAG=1:RETURN
2540 IF L$="Q" THEN PRINT:RETURN
2550 ADD$=ADD$+L$:IF LEN(ADD$)=4 THEN PRINT ELSE 2510
2560 PAGE$=LEFT$(ADD$,2):BYTE$=RIGHT$(ADD$,2)
2570 CON$=PAGE$:GOSUB 3110:IF SUM=-1 THEN 2500
2580 PAGE=SUM-BIAS
2590 IF PAGE>PAGES-1 OR PAGE<0 THEN PRINT"<<<<< OUT OF RANGE >>>>>":GOTO 2500
2600 CON$=BYTE$:GOSUB 3110:IF SUM=-1 THEN 2500
2610 BYTE=SUM
2620 RETURN
2630 REM  ==========================================
2640 REM WRITE TO ARRAY — BYTE BY BYTE
2660 XFLAG=0:DATUM$=" ":PRINT"<<< WRITE MODE >>>  ENTER DATA IN HEXADECIMAL (hh) —> ";
2670 D$=INPUT$(1):PRINT D$;
2680 IF D$>="a" AND D$<="z" THEN D$=CHR$(ASC(D$) AND 95)
2690 IF D$="H" THEN HFLAG=1:RETURN
2700 IF D$="Q" THEN PRINT:RETURN
2710 IF D$="X" THEN XFLAG=1:DATUM$=" ":GOTO 2670
2720 DATUM$=DATUM$+D$:IF LEN(DATUM$)<>2 THEN 2670
2730 PRINT:CON$=DATUM$:GOSUB 3110:DATUM=SUM
2740 IF SUM=-1 THEN 2660
2750 IF (ARRAY(PAGE*256+BYTE) AND 255)<>255 AND XFLAG=0 THEN 2830
2760 DATUM=DATUM OR 256:REM                          TAG LOCATION AS WRITTEN TO
2770 ARRAY(PAGE*256+BYTE)=DATUM:REM                  WRITE DATUM TO ARRAY
2780 GOSUB 2320:REM                                  DISPLAY WRITE TO ARRAY
2790 IF BYTE=255 AND PAGE=PAGES-1 THEN RETURN
2800 GOSUB 2440:REM                                  INCREMENT ADDRESS
2810 GOSUB 2320:REM                                  DISPLAY NEXT LOCATION
2820 GOTO 2660
```

(continued)

```
2830 PRINT:PRINT"<<<<<< ILLEGAL WRITE TO PREVIOUSLY PROGRAMMED LOCATION >>>>>>"
2840 RETURN
2850 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
2860 REM HELP ROUTINE
2880 GOSUB 2250:REM                                    CLEAR SCREEN
2890 PRINT"To initialize the program you should enter the beginning page"
2900 PRINT"address of the EPROM to be programmed. This value is used when"
2910 PRINT"printing to the screen and as a bias value in the write modes."
2920 PRINT"The following single-letter commands are used to control the"
2930 PRINT"modes of the EPROM programmer:":PRINT
2940 PRINT"     (I) INITIALIZE BASE-PAGE ADDRESS — base address is ";BIAS$;"00"
2950 PRINT"     (V) VERIFY ERASURE"
2960 PRINT"     (N) DISPLAY NEXT BYTE"
2970 PRINT"     (P) DISPLAY PREVIOUS BYTE"
2980 PRINT"     (O) OFFSET TO NEW PAGE AND BYTE"
2990 PRINT"     (L) LOAD ARRAY FROM DISK"
3000 PRINT"     (S) SAVE ARRAY ON DISK"
3010 PRINT"     (W) ENTER BYTE WRITE MODE                  (use Q or H to exit, X to edit)"
3020 PRINT"     (D) HEXADECIMAL DUMP TO SCREEN"
3030 PRINT"     (B) ENTER 'BURN EPROM' MODE"
3040 PRINT"     (H) ENTER HELP MODE                        (from any input statement)"
3050 PRINT"     (E) EXIT PROGRAM"
3060 RETURN
3070 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
3080 REM ***** CONVERT HEXADECIMAL TO DECIMAL *****
3090 REM ENTER WITH HEXADECIMAL STRING IN CON$, EXIT WITH DECIMAL VALUE IN SUM
3110 SUM=0
3120 FOR I=1 TO LEN(CON$)
3130   X=ASC(MID$(CON$,(LEN(CON$)+1-I),1))
3140   IF X<48 OR X>70 THEN SUM=-1:I=LEN(CON$):GOTO 3190
3150   IF X>57 AND X<65 THEN SUM=-1:I=LEN(CON$):GOTO 3190
3160   IF X<64 THEN X=X-48 ELSE X=X-55
3170   SUM=SUM+(X*16^(I-1))
3180   IF SUM>255 OR SUM<0 THEN SUM=-1
3190   IF SUM=-1 THEN PRINT"<<<<< INPUT ERROR >>>>>"
3200 NEXT I:RETURN
3210 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
3220 REM WRITE A BYTE
3240 WPAGE=PAGE OR 128:REM                             SET WRITE PAGE (W/R=1)
3250 PRINT #3,CHR$(WPAGE);:REM                         SEND WRITE PAGE
3260 PRINT #3,CHR$(BYTE);:REM                          SET WRITE BYTE
3270 PRINT #3,CHR$(DATUM);:REM                         DATA TO WRITE
3280 PRINT #3,CHR$(STROBE);:REM                        WRITE STROBE
3290 FOR DEL=1 TO DELAY:NEXT DEL:REM                   WRITE DELAY
3300 RETURN
3310 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
3320 REM READ A BYTE
3340 PRINT #3,CHR$(PAGE);:REM                          SET READ PAGE (W/R=0)
3350 PRINT #3,CHR$(BYTE);:REM                          SET READ BYTE
3360 PRINT #3,CHR$(DUMMY);:REM                         DUMMY DATA SENT
3370 PRINT #3,CHR$(STROBE);:REM                        READ STROBE
3380 RDATA=ASC(INPUT$(1,#3)):REM                       INPUT DATA
3390 RETURN
3400 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
3410 REM VERIFY ERASURE
3430 PRINT:PRINT"VERIFYING THAT EPROM IS ERASED":PRINT
3440 BYTE=0:PAGE=0
3450 FOR PAGE=0 TO PAGES-1:V$=" OK"
3460   FOR BYTE=0 TO 255
3470     IF (ARRAY(PAGE*256+BYTE) AND 255)=255 THEN 3490
3480     V$="<<<<<< NOT ERASED >>>>>>"
3490   NEXT BYTE:PRINT"PAGE";PAGE;V$
3500 NEXT PAGE
3510 BYTE=0:PAGE=0:RETURN
3520 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
3530 REM DUMP TO SCREEN
3550 GOSUB 2250
3560 FOR LN=1 TO 16
3570   DPAGE$=RIGHT$("0"+HEXADECIMAL$(BIAS+PAGE),2)
3580   DBYTE$=RIGHT$("0"+HEXADECIMAL$(BYTE),2)
3590   PRINT USING"\\";DPAGE$;DBYTE$;": ";
3600   FOR D=1 TO 16
3610     DDATA$=RIGHT$("0"+HEXADECIMAL$((ARRAY(PAGE*256+BYTE) AND 255)),2)
3620     PRINT USING"\\";DDATA$;
3630     IF PAGE=PAGES-1 AND BYTE=255 THEN D=16:LN=16
3640     GOSUB 2440:IF BYTE MOD 16=0 THEN PRINT:D=16
3650   NEXT D
3660 NEXT LN:PRINT:PRINT
3670 IF PAGE=PAGES-1 AND BYTE=255 THEN PRINT"<<<<<< END OF EPROM >>>>>":RETURN
3680 PRINT"ENTER (C) TO CONTINUE OR (Q) TO EXIT DUMP —> ";:IKEY$=INPUT$(1)
3690 IF IKEY$>="a" AND IKEY$<="z" THEN IKEY$=CHR$(ASC(IKEY$) AND 95)
```

```
3700 PRINT IKEY$:PRINT:IF IKEY$ = "C" THEN 3560
3710 IF IKEY$ = "H" THEN HFLAG = 1:RETURN
3720 IF IKEY$ = "Q" THEN RETURN ELSE 3680
3730 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
3740 REM SET BIAS ADDRESS
3760 GOSUB 2250
3770 BIAS$ = " ":PRINT:PRINT"ENTER BASE-PAGE ADDRESS IN HEXADECIMAL (hh) —> ";
3780 B$ = INPUT$(1):PRINT B$;
3790 IF B$ > = "a" AND B$ < = "z" THEN B$ = CHR$(ASC(B$) AND 95)
3800 IF B$ = "H" THEN HFLAG = 1:RETURN
3810 IF B$ = "Q" THEN PRINT:RETURN
3820 BIAS$ = BIAS$ + B$:IF LEN(BIAS$) < > 2 THEN 3780
3830 PRINT
3840 CON$ = BIAS$:GOSUB 3110:BIAS = SUM:PRINT:PRINT:IF SUM = – 1 THEN 3770
3850 PAGE = 0:BYTE = 0:BP$ = BIAS$ + "00":GOSUB 2250:RETURN
3860 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
3870 REM READ EPROM TO ARRAY
3890 PAGE = 0:BYTE = 0:GOSUB 2250
3900 GOSUB 3340
3910 ARRAY(PAGE*256 + BYTE) = RDATA:IF BYTE = 0 THEN PRINT"READING PAGE";PAGE
3920 BYTE = BYTE + 1:IF BYTE = 256 THEN PAGE = PAGE + 1:BYTE = 0
3930 IF PAGE < = PAGES – 1 THEN 3900
3940 PRINT:PAGE = 0:BYTE = 0:RETURN
3950 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
3960 REM WRITE ARRAY TO EPROM
3980 GOSUB 2250
3990 PRINT"< < < < < < BURN ALL PROGRAMMED BYTES ?? > > > > > >"
4010 PRINT"TYPE (Y) TO PROGRAM EPROM"
4020 PRINT"(Q) TO RETURN TO PROGRAM"
4030 PRINT"(H) TO DISPLAY HELP MENU"
4040 PRINT"(N) TO RETURN TO PROGRAM FROM 'BURN' MODE"
4050 PRINT"TO ABORT PROGRAM IN 'EXIT' MODE."
4060 PRINT:PRINT"ENTER SELECTION —> ";:IKEY$ = INPUT$(1)
4070 PRINT IKEY$
4080 IF IKEY$ > = "a" AND IKEY$ < = "z" THEN IKEY$ = CHR$(ASC(IKEY$) AND 95)
4090 IF IKEY$ = "N" THEN RETURN
4100 IF IKEY$ = "H" THEN HFLAG = 1:RETURN
4110 IF IKEY$ = "Q" THEN PRINT:RETURN
4120 IF IKEY$ < > "Y" THEN 3990
4130 FOR ADD = 0 TO DSIZE
4140   DATUM = ARRAY(ADD):IF DATUM < 256 THEN 4190
4150   DATUM = DATUM AND 255:BYTE = ADD MOD 256:PAGE = (ADD – BYTE)/256
4160   PRINT "BURNING ";:GOSUB 2320
4170   GOSUB 3240:GOSUB 3340
4180    IF RDATA < > DATUM THEN PRINT "< < < < < < DATA NOT VERIFIED > > > > > >"
4190 NEXT ADD
4200 PRINT:BYTE = 0:PAGE = 0:RETURN
4210 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
4220 REM SAVE ARRAY IN DISK FILE
4240 GOSUB 2250:PRINT"THE DISK FILE CREATED HERE WILL CONTAIN ALL THE DATA"
4250 PRINT"PRESENTLY CONTAINED IN YOUR EPROM MEMORY IMAGE AND"
4260 PRINT"WILL BE ASSIGNED THE FILE EXTENSION 'PRM'."
4270 PRINT"THE FOLLOWING IS A LIST OF EXISTING DISK FILES WITH"
4280 PRINT"THE FILE EXTENSION '.PRM'.":PRINT:PRINT
4290 FILES "*.PRM":PRINT:PRINT
4300 INPUT"ENTER THE FILENAME OF YOUR NEW DISK FILE —> ",FILENAME$
4310 IF FILENAME$ = "H" OR FILENAME$ = "h" THEN HFLAG = 1:RETURN
4320 IF FILENAME$ = "Q" OR FILENAME$ = "q" THEN RETURN
4330 OPEN "O",#1,FILENAME$ + ".PRM"
4340 FOR I = 0 TO DSIZE – 1:PRINT #1,(ARRAY(I) AND 255);
4350   IF I MOD 256 = 0 THEN PRINT "SAVING PAGE";I/256
4360 NEXT I:CLOSE #1:RETURN
4370 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
4380 REM LOAD ARRAY FROM DISK
4400 GOSUB 2250:PRINT:PRINT"THE FOLLOWING IS A LIST OF FILENAMES WITH THE FILE"
4410 PRINT"EXTENSION '.PRM'":PRINT:PRINT
4420 FILES "*.PRM":PRINT:PRINT
4430 INPUT"ENTER A FILENAME FROM THE LIST ABOVE —>",FILENAME$
4440 IF FILENAME$ = "H" OR FILENAME$ = "h" THEN HFLAG = 1:RETURN
4450 IF FILENAME$ = "Q" OR FILENAME$ = "q" THEN RETURN
4460 OPEN "I",#1,FILENAME$ + ".PRM"
4470 FOR I = 0 TO DSIZE – 1:INPUT #1,DATUM
4480   IF I MOD 256 = 0 THEN PRINT "LOADING PAGE";I/256
4490   IF DATUM = 255 OR DATUM = (ARRAY(I) AND 255) THEN 4560
4500   IF ARRAY(I) < > 255 THEN 4520
4510   ARRAY(I) = DATUM OR 256:GOTO 4560
4520   PRINT"< < < < < < ILLEGAL INPUT DATA FROM FILE > > > > > >"
4530   PRINT"< < < < < < ATTEMPT TO WRITE OVER PROGRAMMED LOCATION > > > > > >"
4540   PRINT"< < < < < < PROGRAM HAS BEEN ABORTED > > > > > >"
```

(continued)

```
4550   CLOSE#1:END
4560 NEXT I:CLOSE #1:RETURN
4570 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
4580 REM DISK-ERROR ROUTINE
4600 IF ERR=53 AND ERL=4290 THEN PRINT"NO PRM FILES":RESUME 4300
4610 IF ERR=53 AND ERL=4420 THEN PRINT"NO PRM FILES":GOTO 4670
4620 IF ERR=53 AND ERL=4460 THEN PRINT"UNKNOWN FILE":GOTO 4670
4630 IF ERR=61 THEN PRINT "DISK FULL":GOTO 4670
4640 IF ERR=57 THEN PRINT"RESET EPROM PROGRAMMER":GOTO 4670
4650 IF ERR=67 THEN PRINT"UNKNOWN FILENAME, DON'T TYPE '.PRM'":GOTO 4670
4660 CLOSE#1:PRINT "UNKNOWN ERROR #";ERR;"IN LINE #";ERL
4670 PRINT"PRESS ANY KEY TO CONTINUE —> ";:IKEY$=INPUT$(1):PRINT
4680 IF ERR=57 THEN RESUME 0
4690 HFLAG=1
4700 RESUME 2110
4710 ON ERROR GOTO 0
4720 REM  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
4730 REM CONFIGURATION ROUTINE
4750 DATA 255,255,196,255,196,255,196,255,255,196,255,196
4760 DATA 026,196,255,255,255,196,255,196,196,255,255,196
4770 DATA 196,255,196,196,255,255,196,255,196,255,255,255
4780 DATA 196,255,196,196,255,255,196,255,196,255,196,255
4790 IF ESIZE=1 THEN RESTORE 4750
4800 IF ESIZE=2 THEN RESTORE 4760
4810 IF ESIZE=3 THEN RESTORE 4770
4820 IF ESIZE=4 THEN RESTORE 4780
4830 LOCATE 1,22:PRINT "JUMPER CONFIGURATION"
4840 LOCATE 3,30:PRINT CHR$(201);CHR$(205);CHR$(205);CHR$(187)
4850 FOR I=4 TO 15
4860   LOCATE I,30:PRINT CHR$(199);"  ";CHR$(182);"J";I-3
4870 NEXT I
4880 LOCATE 16,30:PRINT CHR$(200);CHR$(205);CHR$(205);CHR$(188)
4890 FOR I=4 TO 15
4900   READ JUMPER
4910   LOCATE I,31:PRINT CHR$(JUMPER);CHR$(JUMPER)
4920 NEXT I
4930 LOCATE 4,38
4940 IF ESIZE=2 THEN PRINT"NOTE: INSTALL J1 FOR 2732A EPROMs"
4950 LOCATE 18,20:PRINT "If jumpers are not properly configured"
4960 LOCATE 19,20:PRINT "shut off programmer and set jumpers,"
4970 LOCATE 20,20:PRINT "then turn programmer back on."
4980 LOCATE 22,20:PRINT "Press any key to continue —> ";
4990 A$=INPUT$(1):RETURN
```

created with a SAVE command in the program can also be used to enter the data.

A help routine is provided in the program to assist the user during the operation of the programmer. It consists of a menu that contains all the choices available in the driver program. The routine can be entered from any location in the program by typing the letter H. A screen-dump routine and an EPROM erasure-verification routine are also provided.

In Conclusion

The serial-port EPROM programmer isn't designed for volume programming. It's intended to be a cost-effective, transportable programmer that doesn't become outmoded with each new computer and system bus. You'll also find, cleverly embedded in every programming cycle, enough time for you to take a well-deserved coffee break.

Circuit Cellar Feedback

This month's feedback begins on page 393.

Next Month

I've always been intrigued by home control and electronic messaging. In March, I'll tackle the subject in earnest, beginning with a Touch-Tone Interactive Message System. ■

Special thanks to Larry Bregoli for his software expertise.

Editor's Note: Steve often refers to previous Circuit Cellar articles. Most of these past articles are available in reprint books from BYTE Books, McGraw-Hill Book Company, POB 400, Hightstown, NJ 08250.

Ciarcia's Circuit Cellar, Volume I covers articles that appeared in BYTE from September 1977 through November 1978. *Volume* II covers December 1978 through June 1980. *Volume* III covers July 1980 through December 1981. *Volume* IV covers January 1982 through June 1983.