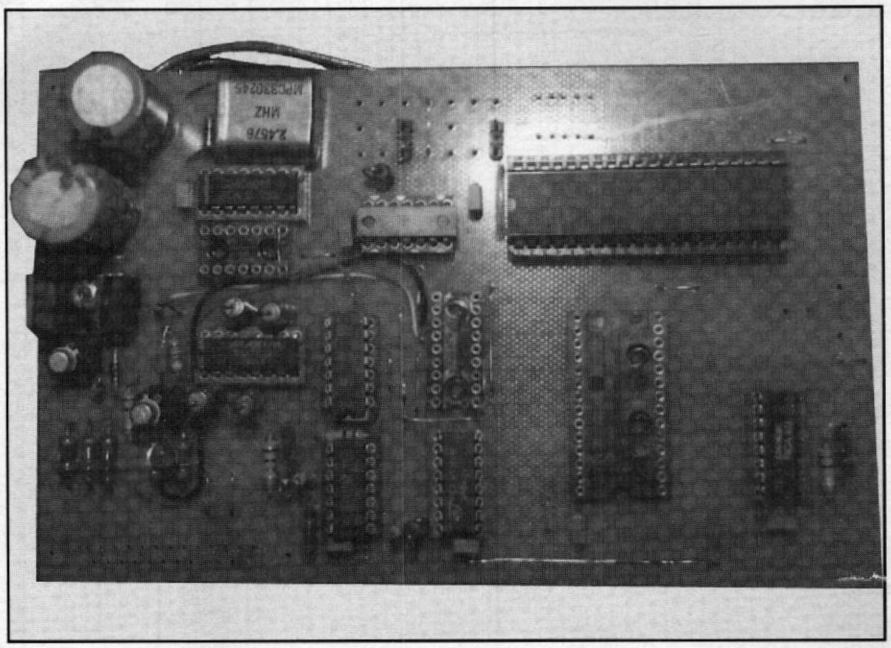
By Gene Francisco

An Economy EPROM Programmer

Off-the-shelf components and a serial connection to your computer easily justify the cost of owning this must-have accessory

Nowadays, more and more people are finding it a must to be able to program their own EPROMs. However, many such people find the high cost of a commercial programmer difficult to justify. If you're in this category, there's a way out that won't break the bank: build the Economy EPROM Programmer described here. Because all of the intelligence for the Programmer resides in the host computer with which it's used, our Economy EPROM Programmer eliminates the need for a costly processor, firmware and memory of its own.

Though a programmer in most setups needs direct contact with the host PC's bus, the Economy EPROM Programmer doesn't even require that you open your computer's system unit because it simply connects to any computer via a standard RS-232 serial port and is ready to go. All power-supply components mount on the same printed-circuit board that accommodates the main circuit components. The only off-board component is a standard plug-in 12-volt ac modular transformer. And this project no "special" or difficult-to-obtain components.



About the Circuit

The protocol for the Economy EPROM Programmer requires that the host computer send a byte of data and then wait for a byte to be returned from the Programmer. Depending on the direction of information flow, one or the other of these bytes is a dummy, used only to signal receipt of or request for a meaningful byte. Half the pairs of bytes carry data. The other half carry control information.

The RS-232 cable that connects the Programmer to the host computer can be up to 3 feet in length, which can deliver a data-transfer rate of up to 9,600 baud. At this speed, each byte requires about 1 ms to be transmitted. Four bytes are exchanged for each byte of meaningful data. Compare this to the 50-ms time required to program an EPROM byte, and the time required for handshake overhead becomes trivial. Reading an EPROM is equally efficient. For example, it takes only 32 seconds to read the entire contents of a 2764 EPROM.

The only complex part of the Programmer's circuit, shown in Fig. 1, is UART *IC1*, used here for data communication between Programmer and host computer. This IC notifies the host when a complete byte has been received. A simple monostable multivibrator causes the UART to send a return byte. A toggle flip-flop at the output of the monostable multivibrator causes alternate bytes to be treated as data or control pairs. Control pairs have only one function: advancing the counter that addresses the EPROM.

Read and write are manual functions that are completely dependent upon the user, which simplifies the protocol. Assuming the reset has been strobed, the address presented at the EPROM is now 0. Assert the read, and a dummy byte is received from the host, the UART sets RDA, the monostable multivibrator presents its delay and then strobes TDS. This sequence causes the UART to send the data byte of the present EPROM address to the host. Simultaneously, RDA is reset through RDAR.

The pulses on RDA toggle 4013 flip-flop *IC4*. The RC circuit and Schmitt input of *IC2* causes a delay that allows the programming pulse to fall before the address changes. The falling edge of the *IC4* advances the counter string composed of *IC2* and *IC3A*. As even-numbered dummy bytes are received, addresses presented to the EPROM advance through all possible locations. As odd-numbered dummy bytes are received, the data associated with the previous addresses is presented. In this fashion, the entire EPROM is read.

Writing is similar to reading, except that the write operation is asserted, instead of read operation. The UART then outputs data to the EPROM. Now the

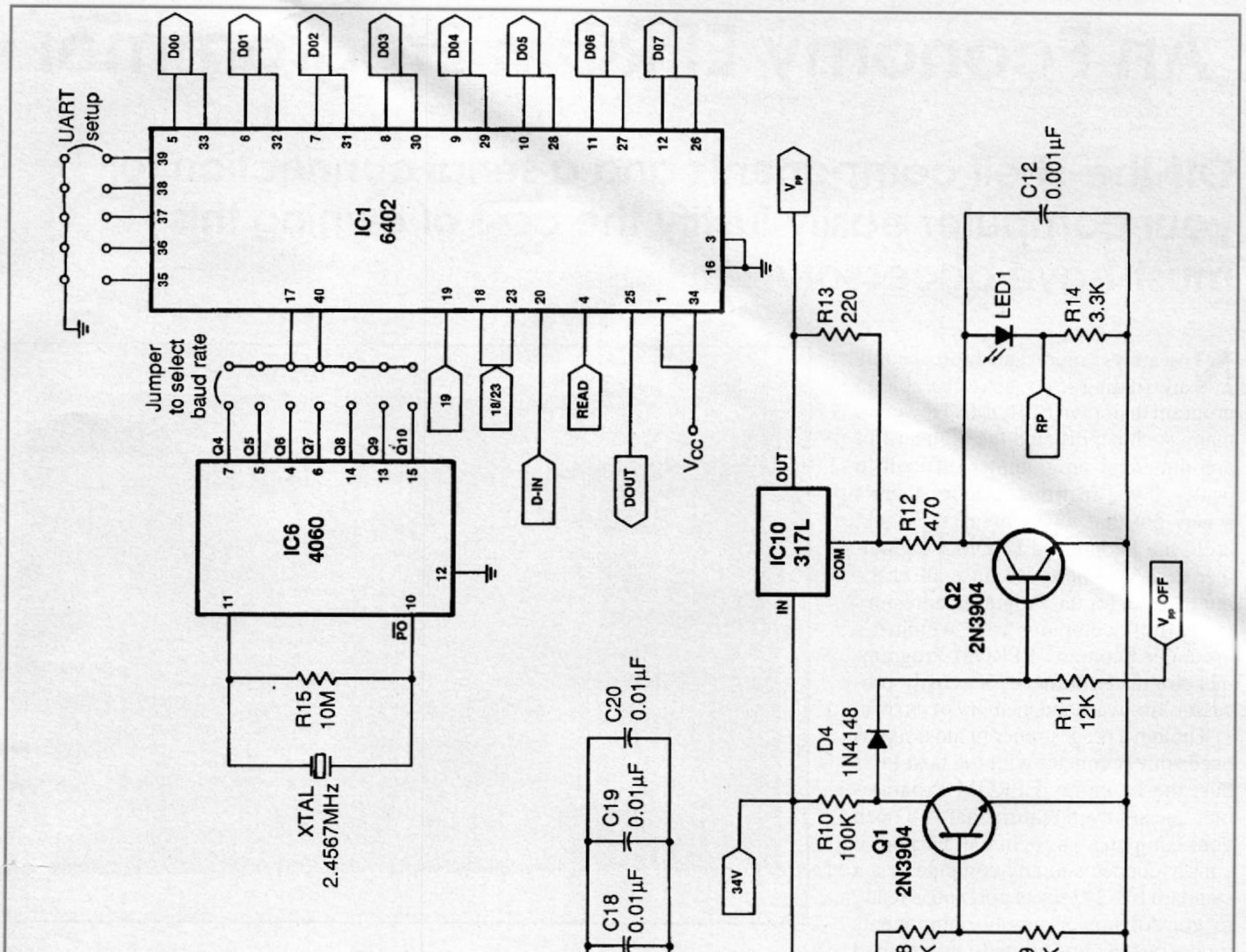
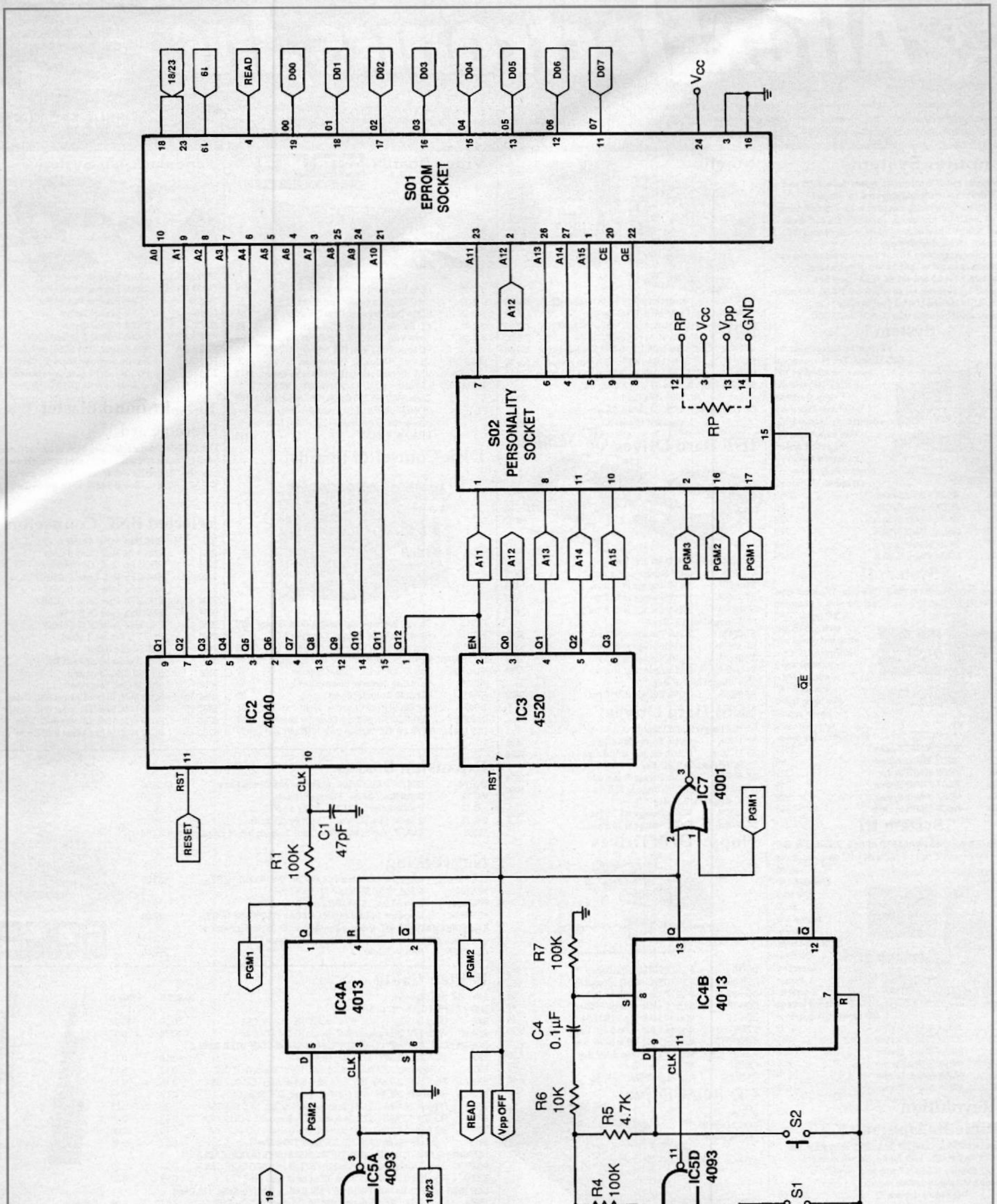


Fig. 1. Complete schematic diagram of Economy EPROM Programmer circuit, minus its power supply.


S01 EPROM SOCKET
S02 PERSONALITY SOCKET
IC2 4040
IC3 4520
IC4A 4013
IC4B 4013
IC5A 4093
IC5D 4093
IC7 4001
R1 100K
R2 12K
R3 100K
R4 100K
R5 4.7K
R6 10K
R7 100K
C1 47pF
C2 47pF
C3 0.047µF
C4 0.1µF
D1 1N4148
S1
S2
RP
VCC
Vpp
GND
RESET
READ
VppOFF
PGM1
PGM2
PGM3
18/23
19
D00
D01
D02
D03
D04
D05
D06
D07
A11
A12
A13
A14
A15
CE
OE

(B)

PARTS LIST

Semiconductors
D1,D4—1N4148 switching diode
D2,D3—1N4001 power rectifier diode
LED1—Red 20-mA light-emitting diode
IC1—6402 UART
IC2—CD4040 counter
IC3—4520 dual binary up/down counter
IC4—CD4013 dual flip-flop
IC5—CD4093 quad two-input NAND gate
IC6—CD4060 14-stage counter
IC7—CD4001 quad two-input NOR gate
IC8—MAX-232 RS-232 transceiver
IC9—7805 fixed +5-volt regulator
IC10—LM317L adjustable voltage regulator
Q1,Q2—2N3904 silicon npn transistor

Capacitors (50-volt)
C1,C3—47-pF ceramic disc
C2—-10,000-pF ceramic disc
C4—0.047-µF ceramic disc
C5 thru C8—22-µF electrolytic
C9—470-µF electrolytic
C10—1,000-µF, 35-volt electrolytic
C11—10-µF electrolytic
C12—0.01-µF ceramic disc
C13 thru C19—0.1-µF ceramic disc

Resistors (1/4-watt, 5% tolerance)
R1,R3,R6,R11,R16—100,000 ohms
R2—4,700 ohms
R4—10,000 ohms
R5,R10,R12—12,000 ohms
R8—10 megohms
R9—39,000 ohms
R13—470 ohms
R14—220 ohms
R17—4,700 ohms
R15—3,360 ohms, 1% tolerance

Miscellaneous
S1,S2—Spst momentary-action pushbutton switch
SO1—28-pin ZIF socket
T1—12-volt, 3-ampere dc plug-in power module
XTAL—2.4576-MHz crystal
Printed-circuit board; suitable enclosure; sockets for all DIP ICs (optional);18-pin personality socket; 18-pin headers for personality module; DB-25 connector; software; machine hardware; hookupwire; solder; etc.

Note: Software for IBM PC/compatible computers for operation on serial ports COM1 and COM2 and a program for comparing two bi- nary files are available for $25 from Gene Francisco, 10266 N. 29 St., Tampa, FL 33612.

host must send the data to be written, rather than just dummy bytes to advance the addresses.

Data sent back during programming from *IC1* to the host computer functions only as a semaphore that indicates receipt of a byte. In addition to advancing the counters, the *IC4* flip-flop controls the program-enable pin of *IC1*, which must be asserted for 50 ms per byte programmed, as required by the host computer's software.

The remaining circuitry consists of a bit-rate generator, interlocking electronic read/program switch, logic-controlled power supply, line interface and personality headers to permit many different types of EPROMs to be programmed. Circuitry is also required to generate three different types of programming pulses in accord with the requirements of various types of EPROMs. For example, 2732 and 27512 EPROMs use the same pin for programming voltages and enable inputs, which is achieved by active components on the personality headers. These need not be included if they aren't needed.

The interlocking programming switch uses a spare section of *IC5* and the spare half of IC4. This system isn't absolutely necessary because just toggle and pushbutton switches would do the job. The interlock is intended to prevent accidents in three ways. The 4093 and the RC circuit on the set pin of the 4013 assert a power-on reset so that the circuit always powers up ready to read, not program. This eliminates the possibility of damaging an EPROM if it's plugged in at turn-on time. Secondly, to change over to programming mode, a two-finger operation is required. This requires some thought and should diminish the chance of accidental damage. Thirdly, momentary-action push-to-make switches are inexpensive and simple to implement and, since the power supply provides a warning indicator, malfunction is unlikely and can't be disguised, as it might be with a toggle switch.

The power supply for this circuit is somewhat unusual. It uses a single transformer winding to provide a variable programmable supply that's adjustable between +5 and +25 volts and a fixed supply that provides +5 volts.

The circuitry around adjustable regulator *IC10* needs some explanation. Resistor *R11*, which can be made up of two resistors in series, allows standard resistor values for R_P (in the personality header) to set the programming potentials at 25, 21 and 12.5 volts. Regardless of the R_P value, the *LED* carries a constant current, given by 1.2 volts/ 220 ohms, or 5.45 mA, whenever V_{PP} off is low. Thus, the brightness of the *LED* doesn't depend on the type of EPROM being programmed. Transistor *Q2*, controlled by V_{PPOFF}, switches off the *LED* and reduces V_{PP} to the TTL high level set by *R12*, as required by most EPROMs when they're read. The purpose of the additional transistor controlled by the 5-volt supply is to ensure that V_{PP} is at the lower level whenever the 5-volt supply goes off. This protects EPROMs from potential destruction.

An addition favorable feature of the V_{PP} supply is that when *IC10* cyclically overheats as a programming current of more than 30 to 35 mA is drawn, the *LED* flashes to provide a visual indication should a faulty EPROM be detected as it's being programmed.

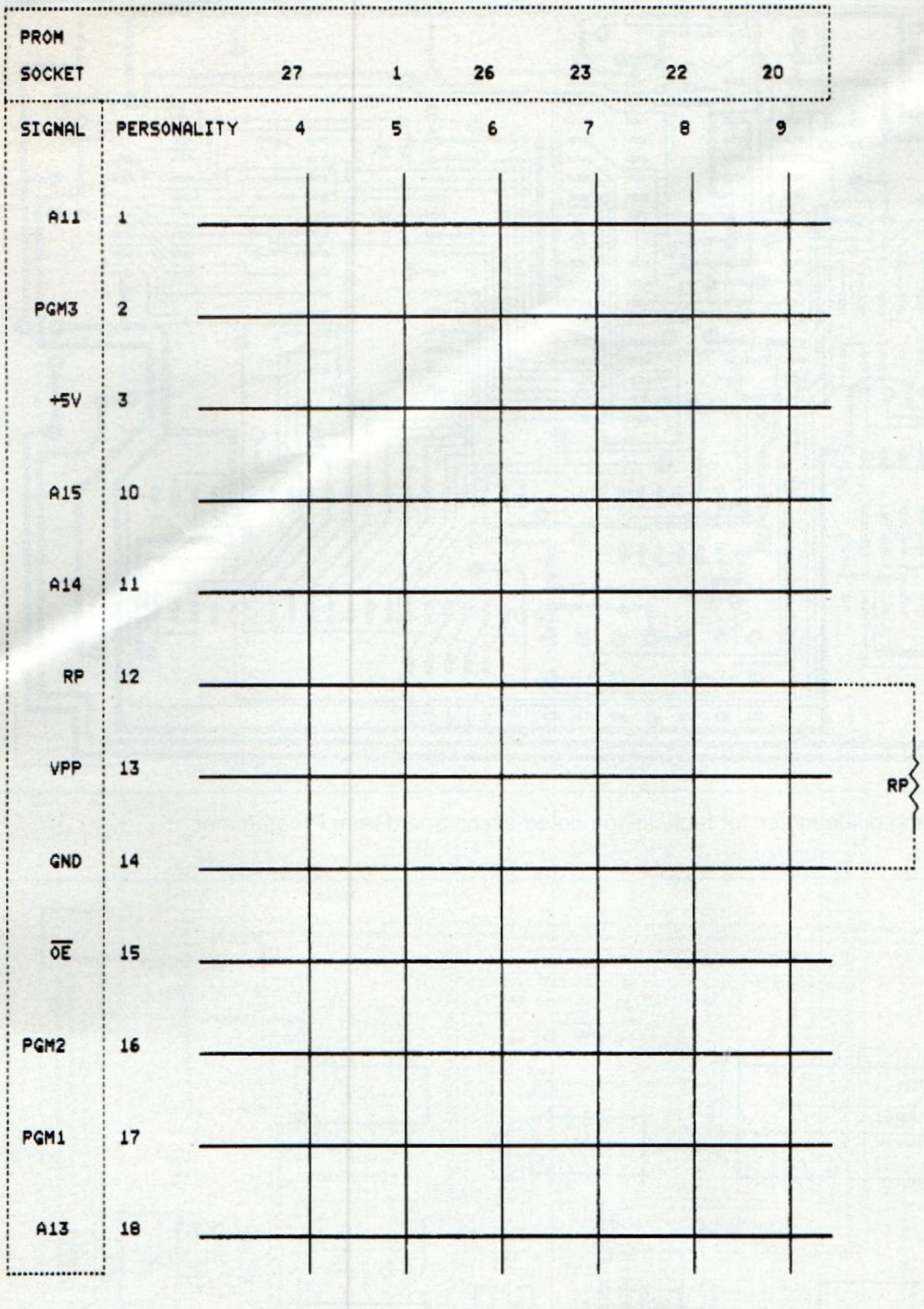
The line interface to the host computer is simple and requires no explanation. Only two active lines are used in the RS-232 connection—DATA IN and DATA OUT—which are always on pins 2 and 3 of a DB-25 connector. Also, pin 7 of the connector is always used as data ground.

Personality header connections for a range of EPROMs are indicated Fig. 2. Also shown is the schematic diagram that pertains to the special personality header required for 27512, 2732 and 2732A type EPROMs.

The Software

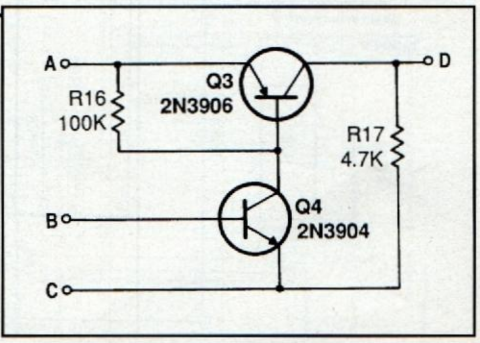
Features of the software break down into three classes: the necessary, the useful and the nice-to-have. Necessary are the features required to write and read EPROMs from and to disk files. Useful features add the ability to display informative screens and warning messages. Nice-to-have features may include provision for progress reports during programming, verifying and copying, differentiation during verification between re-programmable and erasure-required faults and filenames and file-size buffers to eliminate the need to retype the file identification if multiple copying or verification or both are required.

The necessary features are built from the software and include the following: (1) Initialize serial communication to



PENSONALITY RESISTOR (RP)
25V PROMS USE NONE
21V PROMS USE 12K OHMS
12V PROMS USE 1.8K OHMS
USE 1% RESISTOR IF AVAILABLE

2716	2732/2732A	2764/27128	27256	27512
3 TO 6	1 TO 7	1 to 7	1 TO 7	1 TO 7
7 TO 13	2 TO 9	4 to 16	2 TO 9	2 TO 9
9 to 17	3 TO 6	5 to 13	4 TO 11	4 TO 11
8 to 15	A TO 13	6 to 18	5 TO 13	5 TO 10
NO RP	B TO 15	8 to 15	6 TO 18	6 TO 18
	C TO 22	9 to 14	8 TO 15	A TO 13
	D TO 14			B TO 15
				C TO 22
				D TO 14



PERSONALITY WIRING

Fig. 2. Details for personality module and a schematic for wiring a module for use with 27512, 2732 and 2732A EPROMs.

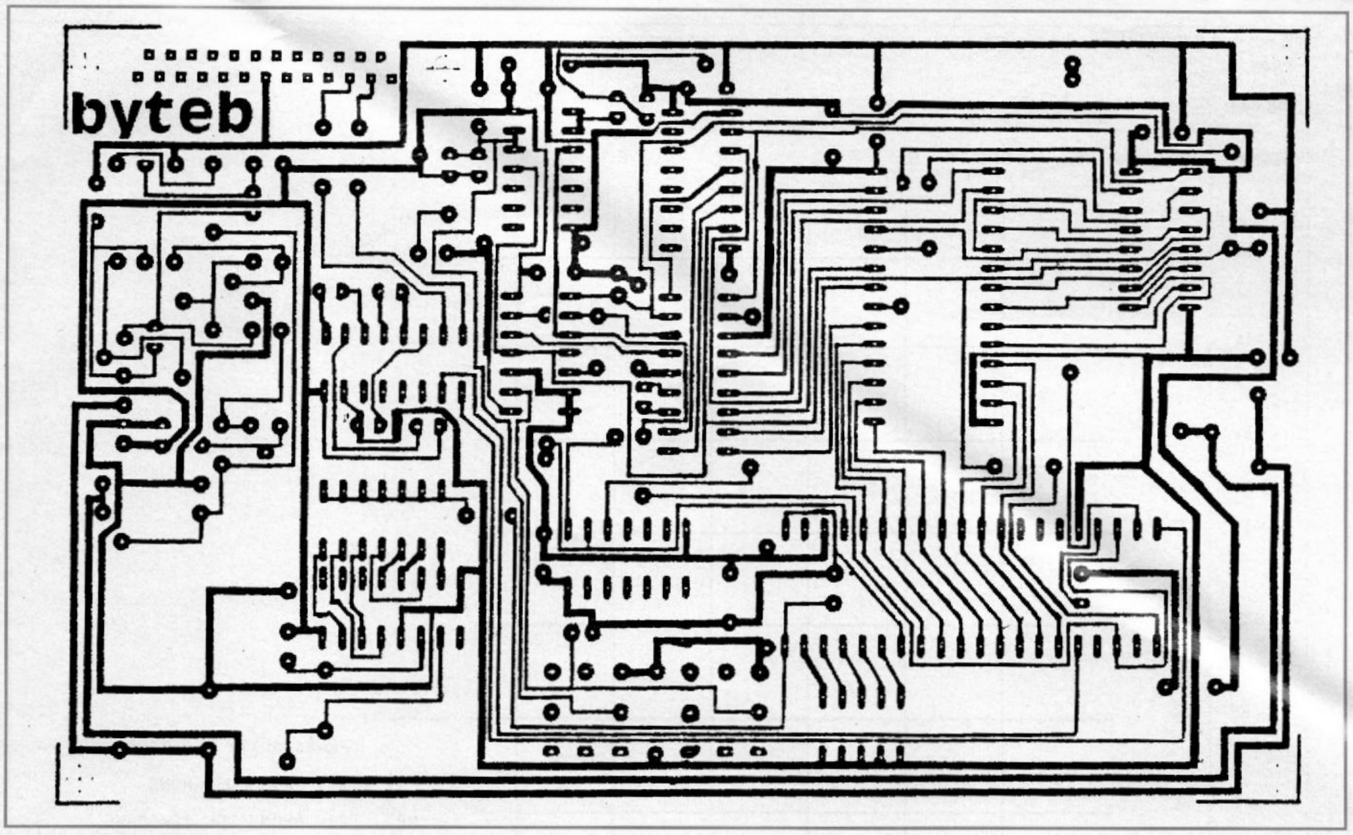
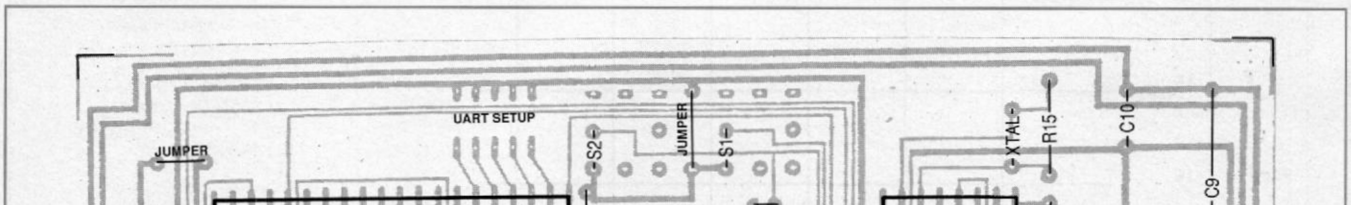


Fig. 3. Actual-size etching-and-drilling guide to use for fabricating printed-circuit board from Programmer.



NOTE:
This is the view from the *trace* side of the board.

Fig. 4. Wiring guide for pc board.

9,600 baud, no parity, two stop bits and eight data bits.
(2) Open a disk file for reading or writing, and close it again.
(3) Flush the computer's and/or UART's serial receive buffer.
(4) Output an eight-bit character through the serial port with no handshaking.
(5) Wait for and eight-bit character to arrive through the serial port and input it with no handshaking.
(6) Do nothing for 50 ms (±5 ms).

The two essential functions of the EPROM Programmer are transfer of EPROM contents to and from disk files. These use the following program flows.

For reading, initialize, flush the buffer, open a disk file into which to read and then repeat the following as many times as other are bytes to read:
(1) Send a dummy byte (say 0).
(2) Input a byte and store it as next in the file.
(3) Send another dummy byte.
(4) Input a byte and discard it.
After all bytes are read close the input file.
For writing, initialize, flush the buffer, open a disk file from which to write and then repeat the following as many time as there are bytes to write:
(1) Fetch the first/next byte from the file and output it.
(2) Unless the fetched byte is FF16h, delay 50 ms.
(3) Input a byte and discard it.
(4) Output the fetched byte again.
(5) Input a byte and discard it.
(6) Finally, inform the user writing is done and close the output file.

Output and input operations with no handshake requires an interface directly to the hardware of the UART, as operating-system calls (for example under MS-DOS) may scan DTR or other signals before sending or receiving serial bytes. The alternative to tracking the trouble with software (not recommended) is to wire a special DB-25 serial plug involving pins 4, 5, 6 and/or 20 and perhaps others with the correct combination of shorting links.

A verify function is very helpful to read sequential bytes from the EPROM but open a file for reading, rather than writing, and comparing the EPROM's contents with sequential bytes from the file and display any differences found. The first of the nice-to-have functions memorizes the identity and size of the last file used to permit the same data to be used again. This feature has certainly saved me enough in time (and typing errors) to have made its implementation worthwhile.

Another function puts asterisks on the screen in blocks that indicate bytes processed, to give a progress report. One asterisk per 64 bytes gives just enough information during programming to give comfort that something is occurring without wasting too much time doing screen writes. It's also useful that the number of bytes to be programmed can be selected to be less than the EPROM's full length, permitting partial programming, where this is suitable. To fully program the popular 2764 EPROM takes eight minutes (less if there are "blank" areas with FFs in them).

A final nice feature is an addition to the verify function that reports whether EPROM erasure is required when a mismatch occurs between a file and the contents of the EPROM. Erasure is required when an EPROM bit is low that should be high. The software reports each such mismatch with a message and an audible signal. This makes checking the suitability of an EPROM for overwriting with a given file a matter of listening, rather than careful watching. Silence during the verify function means overwriting is possible because all mismatched bits are high. ■