Eprom Programmer Mark 2

Originally published by Paul Stenning in ETI, November 1994

A common requirement for the electronics hobbyist today is a cheap straight-forward EPROM Programmer, suitable for occasional use, without the extra features found on many commercial units. The EPROM Programmer presented here is designed to address these requirements, and can be used to program the standard 27 family of devices, from 2764 to 27512. It can be used with any IBM PC compatible computer as the controller and data source.

The 27 series family of EPROMs has now been around for many years, so long in fact that the smaller 24 pin devices in this family (2708, 2716 and 2732) are almost obsolete and are now more expensive than a 2764. For this reason it was felt to be unnecessary to accommodate these devices in this design since they would significantly increase the complexity.

The design presented here has avoided the usual "Catch 22" situation of requiring a programmed EPROM to make the EPROM programmer work, and the design uses readily available components to reduce the likelihood of obsolescence. The unit is powered from an external PSU, supplying +5V @ 200mA and +12.6V @ 200mA. A suitable design is featured elsewhere in this magazine.

The programmer itself is dumb and is controlled by the host computer via the RS232 serial port (COM1 or COM2). Device selection and operation mode is set by front panel switches.

The accompanying software is available on disk from the author. The software will operate on any PC running MS-DOS or PC-DOS version 3.0 or later and having at least 512K of RAM and one RS232 serial port. A hard disk and a colour monitor are strongly recommended. The software is written in Microsoft QuickBASIC V4.5, and the full source code is supplied for those wishing to enhance or modify it. This source code is also compatible with QBASIC, as supplied with MS-DOS 5.0 and later. You do not need QuickBASIC or QBASIC to use this disk. The disk also contains the software for a matching EPROM Emulator planned for publication next month. A description of the software operation is given later.

The unit may also be suitable for use with other types of home computer having an RS232 serial port, although this has not been tested and no software is available. It will definitely not work with Commodore Amiga computers, due to a peculiarity in the serial port handling.

Please note that the programming algorithm used may not be exactly as specified in some EPROM manufacturer's data sheets. Because of this, the unit cannot be guaranteed to program every device successfully, however no problems have been experienced to date.

How it Works

The circuit diagram is spread over a number of illustrations. Although it may initially look complex, it is in fact relatively straightforward. When a "-" follows a signal name (for example STROBE-), this shows that the line is active low. When a number is followed by an "h" this indicates that the number is hexadecimal.

The RS232 (serial) interface, buffering and clock are shown in figure *. IC3 (6402) is a UART

(Universal Asynchronous Receiver/Transmitter) which basically converts serial data to parallel and vice-versa. The device supports most common serial data formats, in this application it is configured to give eight data bits, one stop bit and no parity checking. The data rate is set by the crystal controlled clock circuit (IC2), which in most cases will be set to 9600 Baud.

IC1 is the RS232 buffer, converting the 5V data from the UART to the +/- 10V RS232 standard, and vice-versa. This device contains a voltage doubler and voltage invertor circuit, producing +/- 10V rails from a single +5V input. The four external capacitors (C1 to C4) are required for these circuits.

A byte of serial data arriving on pin 20 of IC3 will be converted to parallel data, which will appear on pins 5 to 12. Pin 19 will go high, and no further data can be received until pin 18 is taken low momentarily. One gate of IC4 causes this to happen, R3 and C8 create a slight delay giving the STROBE- pulse adequate width for clocking other devices. When pin 23 is pulsed low, the parallel data on pins 26 to 33 is transmitted in serial form from pin 25.

R4, D1, C9 and one gate of IC4 produce the power-on reset pulse for IC3, IC5 and IC6. SW1 allows the circuit to be reset again as required.

The pinouts of the four EPROM types handled by this unit are shown in the table below.

PIN	2764	27128	27256	27512
#1	VPP	VPP	VPP	A15
2	A12	A12	A12	A12
3	A7	A7	A7	A7
4	A6	A6	A6	A6
5	A5	A5	A5	A5
6	A4	A4	A4	A4
7	A3	A3	A3	A3
8	A2	A2	A2	A2
9	A1	A1	A1	A1
10	A0	A0	A0	A0
11	D0	D0	D0	D0
12	D1	D1	D1	D1
13	D2	D2	D2	D2
14	GND	GND	GND	GND
15	D3	D3	D3	D3
16	D4	D4	D4	D4
17	D5	D5	D5	D5
18	D6	D6	D6	D6
19	D7	D7	D7	D7
#20	CE	CE	CE/PP	CE/PP
21	A10	A10	A10	A10
#22	OE	OE	OE	OE/VPP
23	A11	A11	A11	A11
24	A9	A9	A9	A9
25	A8	A8	A8	A8
#26	NC	A13	A13	A13
#27	PP	PP	A14	A14
28	VCC	VCC	VCC	VCC

NC = No Connection
PP = Program Pulse, Negative Going
OE = Output Enable (Active Low)
CE = Chip Enable (Active Low)
VPP = Programming Voltage
VCC = Supply Voltage
GND = Ground (0V)

It can be seen from the above that most of the pins are the same for all devices. Only 5 pins require special attention, these are 1, 20, 22, 26 and 27, and are marked with a "#" next to the pin number.

Pin 26 is not used on the 2764 and A13 on the other devices, since A13 is outside the addressing range of the 2764 there is no problem leaving it connected to Pin 26 when handling these devices.

SW2 selects which signals arrive at the other four pins. This can be followed through with the circuit diagram and the table above - and does not need a tedious description from me. Note that 2764 and 27128 share the same switch position.

SW3 selects either Program or Read mode. With this switch in the centre (Off) position the EPROM receives 5V power only and can be safely removed from the socket (SK2).

Counters IC5 and IC6 are connected to the EPROM address lines, and are arranged to increment each time a byte of data is received. I will describe the full operation sequence for both Read and Program modes shortly.

IC7 is a tri-state buffer and is enabled in Program mode only. In this mode it connects the received data from the UART to the EPROM. Although the 74LS245 is bidirectional, in this application it operates in one direction only because pin 1 is permanently connected to 0V.

OK, it's time to own up! The circuitary around IC10, TR1 and TR2 is an afterthought. If you look at the internal photo of the prototype you will see the Veroboard (don't worry, the current PCB is correct). I missed the fact that 27256 and 27512 EPROMs need a supply of 6V instead of 5V when programming.

Since there are no spare sections on SW3, I am using the section that connects the programming pulse to pin 20. If the unit is switched to Read, or to Prog and 2764/27128, pin 20 of the socket will be low. If it is switched to Prog and 27256 or 27512 it will be high, except when actually programming in which case it will be high with the 1ms negative going programming pulses. This high level turns on TR1 via R17. TR1 discharges C19, which will not charge significatly during the 1ms periods when TR1 is off. This low level keeps TR2 off, allowing R13 and R14 to bias the GND pin of IC10 to about 1.2V, giving an output of about 6.2V. When Pin 20 is low, TR2 is off and TR1 is on. The GND pin of IC10 is therefore connected to 0V, giving an output of 5V.

I chose 6.2V because some devices need 6V while others need 6.25V, in either case the tolerance is +/- 0.25V. The voltage tends to drop slightly on load due to the varying current from the GND pin of IC10, so I tended towards the high end of the acceptable range. Some EPROMs draw 100mA or more, so a 1A regulator IC was chosen in preferance to a 100mA part. IC10 draws it's power from the 12.6V programming supply so this should be suitably rated.

Assume that SW3 is in the Read position. When a byte of data is received, it appears on lines R0-R7, and the Strobe- line pulses low as described previously.

Note that in Read mode the actual value of the byte received is irrelevant, since IC7 is disabled. However the receipt of a byte causes the unit to send a byte, which is what we need.

Since the Prog- line is high, pin 13 of IC4 will be held high by D3, so the Strobe- pulse will have no effect in this section of the circuit. This arrangement of diodes on the input of an invertor gate produces a NOR function.

The Read- signal however is low, so the Strobe- pulse will pass through D5 and the two gates of IC4, arriving on pin 8. PP- is currently high (I'll explain this line in a moment), so the Strobe- signal will appear inverted on pin 4 of IC4. C14, R11 and D9 act as a basic monostable, giving a very brief negative going pulse on the Send- line as IC4 pin 4 goes low.

This Send- pulse causes the UART (IC3) to transmitt the data on it's pins 26-33. These pins are connected to the EPROM data pins, so the data transmitted is that contained in the current EPROM address. RN1 holds the lines high if no EPROM is fitted, since IC3 is a CMOS device.

Pin 24 of IC3 will go low once the data is safely in it's buffer and will go high again once it has been transmitted. This signal is connected to the clock input of IC5 to increment the address.

The next time a byte is received, the data in the next EPROM memory location will be transmitted, and so on. Providing the user presses the Reset button when requested, the software and programmer will remain in step.

We will now look at the Program mode, where the Prog- line is low. IC7 is enabled due to its pin 19 being low. The data received by the programmer will therefore be coupled to the EPROM data lines.

The situation with our diode NOR gates is now reversed, so the Strobe- pulse will appear on pin 12 of IC4, and not pin 8. Another CR monostable circuit (R8, D4 and C16) produces a brief negative going Prg- pulse.

IC8 and IC9 generate the 1 millisecond programming pulse. The lower two gates form a S-R flip-flop. When the Prg- pulse occurs, pin 6 of IC9 will go high, causing pin 12 to go low. This takes the reset pin of IC8 low, so that it can count the pulses on it's clock input. This pulse train (Clock2) is from the crystal controlled clock circuit (IC2) and has a frequency of 19.2KHz (2.4576MHz divided by 128), which equates to a period of 52.1uS. When 19 of these pulses have been counted (19 * 52.1uS = 0.99ms), pin 8 of IC9 will go low, reversing the state of the S-R flip-flop and reseting IC8 again.

Thus a 0.99ms negative going pulse will appear on the PP- line. The specification calls for 1ms +/- 5%, so 0.99ms is fine. This PP- signal will go to the appropriate pin on the EPROM, set by SW2. The rising edge of PP- will produce a negative pulse on Send- (via D7), as before. A byte of data will therefore be sent once the programming operation is complete. This will also cause the address counters to be incremented as described previously. The value of the data sent is not important since it's purpose is to tell the softwsre that the programming step is done - however it will be the same as the data received.

Therefore to program the EPROM, it is simply necessary for the software to send one byte of data, and wait to receive something back before sending the next one.

The use of discrete components to produce other types of logic function from a logic invertor gate may seem odd, but it means I can do the job with one IC instead of four!

Construction

The unit is assembled on a single sided PCB. A number of wire links are required, which should be fitted before any components, since some pass underneath ICs. I would suggest that the resistors are fitted next, followed by the ICs, then the capacitors, then the remaining parts. Fit a link wire between COM and 96 to set the Baud rate to 9600. Fit SIL header strip or Veropins for the off-board connections.

Fit a 28-way IC socket in the EPROM socket position. Plug three more sockets into this, then plug the ZIF (Zero Insertion Force) socket into the top. If this stack feels unsteady use some Araldite or similar to hold it together.

The interwiring is shown in figure *. This should be carried out at this stage, since it is necessary for testing. After testing the board can be fitted into the case.

The connections for both 9 and 25 way serial connectors, use whatever matches the socket on your computer. On the prototype a 9 way D connector (serial) and a 6 way DIN socket (DC input) were fitted to the case.

The rotary switch connections are shown by giving the pin number or letter marked on the switch body. Please take great care with these, since there are 16 wires to each switch and an error could be difficult to track down. I used coloured ribbon cable, and still managed to get one wrong!

Testing

Connect the unit to a 5V and 12.6V supply. If a test meter is to hand, check for about +9V on pin 2 and -9V on pin 6 of IC1 (MAX232).

Connect the unit to the serial port on your PC, and run the program "SER-TEST.EXE" on the software disk. When prompted, type "1" or "2" followed by <Enter> to say which serial port you are using. The program does nothing more exciting than wait for you to enter a 2-digit hex number (followed by <Enter>) and then sends it to the programmer. It then attempts to read back a number, if it's successful it prints the number otherwise it prints "**". To exit just press <Enter> on it's own. All the responses in this section will be shown with quotes ("") around them - just type the number (followed by <Enter>), don't type the quotes.

Connect the programmer to your PC's serial port and a suitable power supply. Switch the programmer to 27512 and Read, and press Reset. Type "00" on the computer, and the software should respond with "FF". Whatever two digit hex number you type now should bring the response "FF", since the unit is reading the EPROM data lines, which with no EPROM will be pulled high by RN1.

We will now pull one of the data lines low, to give a different reading. Fit a piece of wire between pins 9 and 12 of the EPROM ZIF socket. If you type a number now the unit will return "FE". Leave one end of the wire in pin 12, and connect the other end in turn to pins 10, 11, 13, 14, 15, 16 and 17. Type a number in each case and you should get the following responses - "FD", "FB", "F7", "EF", "DF", "BF" and "7F" respectively. Remove the piece of wire, and switch the unit to Off. Now if you type a number the unit will not respond so the software will show "**".

Now switch the programmer to Program. Type "00", and "00" should be returned. Using a logic probe, test meter or oscilloscope, check the logic levels on pins 9, 10, 11, 13, 14, 15, 16 and 17. They should all be low.

Now type "01". The unit should return "01", and pin 9 should now be high, and the others should remain low. Now enter "02", "04", "08", "10", "20", "40" and "80" in turn. In each case the unit should return the number you entered. After each entry, check the logic levels on the data pins, only one should be high in each case - 10, 11, 13, 14, 15, 16 and respectively.

Switch the unit back to Read, and press Reset. Press Enter on it's own to quit the software then run "ADR-TEST.EXE", which is also on the disk. Since the address counters are incremented when a byte is sent, it would take a long time to get the count to 65535 manually! ADR-TEST does it automatically, and pauses at four points to allow you to check the logic levels. Follow the instructions on screen. The table below shows the expected logic levels on the address pins at the four pause points.

Addr Line	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Socket Pin	1	27	26	2	23	21	24	25	3	4	5	6	7	8	9	10
Count	Expected Logic Level															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21845	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
43690	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
65535	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

If these readings are OK, and the preceding tests were also successful you can be fairly confident that the unit is working correctly. The only sections that weren't checked were some of the device and mode switching. However the likelihood of problems here is remote if you were careful with the switch wiring.

If you have a blank EPROM and a suitable hex file available, you can try programming a device using the information given shortly.

The Case

The prototype was constructed in a plastic case, 190mm * 165mm * 68mm, see parts list for details. A suitable overlay for the front panel is shown in figure *. Two photocopies may be taken (enlarge to 162mm * 64mm), one can then be used as a drilling template while the other may fixed to the front panel with clear self-adhesive vinyl.

Cut a rectangular hole in the centre of the top for the ZIF socket, then mount the PCB on suitable spacers. The sockets for DC power input and RS232 input are fitted on the rear panel.

Software

The software for this project is supplied on one 3.5" 720K disk (now available for download - link at top of page). Please note that this disk also contains the software for a matching EPROM Emulator, planned for publication next month.

Since the various programs extend to over 2500 lines of BASIC source code, it would make boring reading to print it in the magazine. A printed listing is not available since it would be more expensive than a disk to produce.

The software is supplied as-is, and Paul Stenning cannot accept any liability for any loss or damage however caused. The source code is supplied so that you may modify the software for your own use only. The software may not be redistributed in either it's original or in a modified form.

A batch file is supplied on the disk to simplify installation. Insert the disk in the drive, type "A:" then "INSTALL", and the batch file will make a \EPROM directory on your drive C:, and copy the software to there. If you do not have a hard disk, make a working copy of the disk using DISKCOPY, then put the original away. Do not write-protect your working copy or the software will not work.

If you are using Windows, suitable icon, PIF and group files are supplied on the disk. Some parts of the software will operate much slower under Windows, particularly the initialisation when the serial port is opened. However it will run in the background (probably very slowly) if you are using 386 Enhanced Mode.

The main software of interest for this project is spread over two programs, "PROGRAM.EXE" and "HEX-CONV.EXE". The first of these is the main control software for the programmer, while the second converts various industry standard hex file formats to and from the EPROM programmer format. Formats supported include Intel, Motorola S-Record (3 variants), Tektronix, RCA Cosmac, Binary Image, ASCII Hex (3 variants) and ASCII Binary (4 variants) - so you should be able to find something suitable!

Additional programs on the disk are "EMULATE.EXE" which controls next months EPROM Emulator, and "SPLIT2.EXE" & "SPLIT4.EXE" which divide Intel hex files into 2 or 4 files for 16 and 32 bit systems respectively. Since the full BASIC source code is given for all of these programs, it would be possible to create one large program containing all the facilities - if someone had more time than me! I would be interested to see any enhanced versions.

When "PROGRAM.EXE" is started, the Device Selection Menu will appear. From here you choose the type of device you will be using, either 2764, 27128, 27256 or 27512. CMOS versions of these devices can also be handled, so if you are using a 27C128 you would choose 27128.

One important point regarding programming voltages. Early 2764 and 27128 devices required a programming voltage of 21V. If the device is marked 27C64, 27C128, 2764A or 27128A it will require 12.5V. Any non-CMOS device without an 'A' suffix will require 21V. If in doubt, try programming with 12.5V first, no damage will occur but the device will not be programmed. If a 12.5V device is programmed with 21V it will be destroyed. 21V devices can be programmed by this unit, but you will need to provide an appropriate voltage, and a heatsink for IC10.

Once you have chosen the device required, the Main Menu will appear. Option 1 allows you to read an EPROM, and save the data to disk. The data is saved and loaded ASCII-Text format which is peculiar to this software. "HEX-CONV.EXE" will convert to and from this format.

When Option 1 is selected you will be told where to set the switches, and to insert the EPROM to be read. You will then be asked for a filename, simply enter 8 alpha-numeric characters - the extension is fixed to .HEX and does not need to be typed. If a file of that name already exists it will be overwritten. Now sit back and wait, the progress will be shown on the screen.

Option 2 allows you to check an EPROM is blank. A blank device will have "FF" in all locations. The operation is the same as above but no filename is needed. Note that if no device is fitted, the blank test will pass since the data lines are pulled high by RN1.

An EPROM can only be erased by exposing the window in the case to Ultra-Violet light. If a proper

eraser is used this will take about 20 minutes. Replacement tubes for these are available for about #10 and will fit into some fluorescent torches - leave out any clear plastic or glass in front of the tube since this will block UV. Be very careful not to look at this light.

I have not yet found a completely successful alternative. A disco type 2' UV fluorescent tube will erase a device in about two days! I have been told that a camera flash gun will erase a device in about 4 flashes, but it did not work with my point-and-shoot camera. A professional flash unit will probably be rather more successful.

Option 3 allows you to program an EPROM from data on disk. Operation is the same as above, but the programming process will take about twice as long as reading. Once the programming is complete you will be asked if you wish to program the device.

Option 4 allows you to verify an EPROM. This checks the EPROM against a file on disk, and is normally done after programming to make sure the programming was successful.

Option 5 allows you to change the EPROM type as previously. Option 6 lets you run the Hex File Convertor program, "HEX-CONV.EXE" - this is detailed below. Option 7 lets you access a DOS Shell, type "EXIT" to return to the programmer. To quit the Programmer, press Escape.

The Hex File Convertor is equally simple to use. When it is started, you get the Load Menu. Choose the file format of the existing file, then enter the filename when requested. As previously, the file extension is .HEX and need not be typed.

Once the file is loaded, the Save Menu will appear. Choose the file format you want, then type the filename. If a file of that name already exists it will be overwritten without warning - so be careful!

The file "HEX-CONV.TXT" gives information on all the file formats, and samples are provided on the software floppy disk, in the \HEX directory.

Conclusion

Elsewhere in this issue you will find a suitable power supply for this unit. It will supply 5V at up to 500mA and 12.6V at up to 250mA. This will also power next month's EPROM Emulator.

Coming next month is a matching EPROM Emulator. This emulates the same range of devices as the programmer, and can save hours programming and erasing EPROMs when doing software development work. It needs a single 5V supply and can be powered from the host circuit (if it can spare 100mA) or from the power supply mentioned above.

Parts

Resistors (0.25W 5% or better)

R1	4M7
R2	2K2
R3	1K0
R4,7,9,14,17	47K
R5,6,8,10,11,16	4K7
R12	100R

R13	470R
R15,18	22K
RN1	10K x 8 Resnet

Capacitors (all 16V or greater)

C1,2,3,4,5	22u
C6,7	22p
C8,16,17	2n2
C9	10u
C10,11,12,13,14,18,20	100n
C15	47u
C19	1u0

Semiconductors

IC1	MAX232
IC2	74HCT4060
IC3	CDP6402 (Maplin QQ04E)
IC4	74HCT14
IC5,8	74HCT4024
IC6	74HCT4040
IC7 74LS245	
IC9 74LS10	
IC10	7805
TR1,2	BC548
D1,2,3,4,5,6,7,8,9	1N4148

Miscellaneous

XT1	2.4576MHz
SK1	9 or 25 way D
SK2	28 way ZIF socket
SK3	DC INPUT
SW1	Push to make
SW2,3	4 pole 3 way rotary

PCB, Case, knobs, wire.

Update

The Harris CDP6402 may be replaced with a Harris HD6402.