JEFF ORTHOBER

EPROM EMULATOR

Build a 2764 fake EPROM project for the IBM PC.

THEY SAY THAT NECESSITY IS THE mother of invention. Sometimes, however, convenience is. This article describes an EPROM emulator for an IBM PC. When you're designing a new project, you can save a lot of time by using a "fake" EPROM instead of going through the usual program and erase cycle.

This project uses static RAM on an IBM PC plug-in card to emulate the 2764. Since the 2764 is an 8K × 8-bit EPROM, the 6264 8K × 8-bit static RAM IC will work fine. To use the emulator, you must download the bits to the RAM. Then plug a cable from the emulator in the EPROM's socket of the project you're developing.

The emulator was built on an IBM PC prototyping board from JDR Microdevices, part number JDR-PR2. (Similar prototyping boards are available from several manufacturers.) Only the necessary buffering and cabling had to be added to complete the project. The IBM PC bus lines that are provided by the prototyping board are lines A0 through A9, AEN, RESET, -MEMR, MEMW, -IOR, and -IOW, and lines D0 through D7. All of the lines are buffered, and the data lines are fully buffered—that is, they are three-stated.

The prototyping board is activated when its I/O read or write line is high, DMA address enable is low, and the on-board comparator's output is high. The comparator on the board defaults to hexadecimal address 300 through 31F. The ports are subdivided eight times by a 3-to-8 decoder, providing lines -SEL0 through -SEL7 on the board. The default settings are shown in Table 1.

TABLE 1

Select Line	I/O Port (hex)
-SEL0	300 - 303
-SEL1	304 - 307
-SEL2	308 - 30A
-SEL3	30C - 30F
-SEL4	310 - 313
-SEL5	314 - 317
-SEL6	318 - 31B
-SEL7	31C - 31F

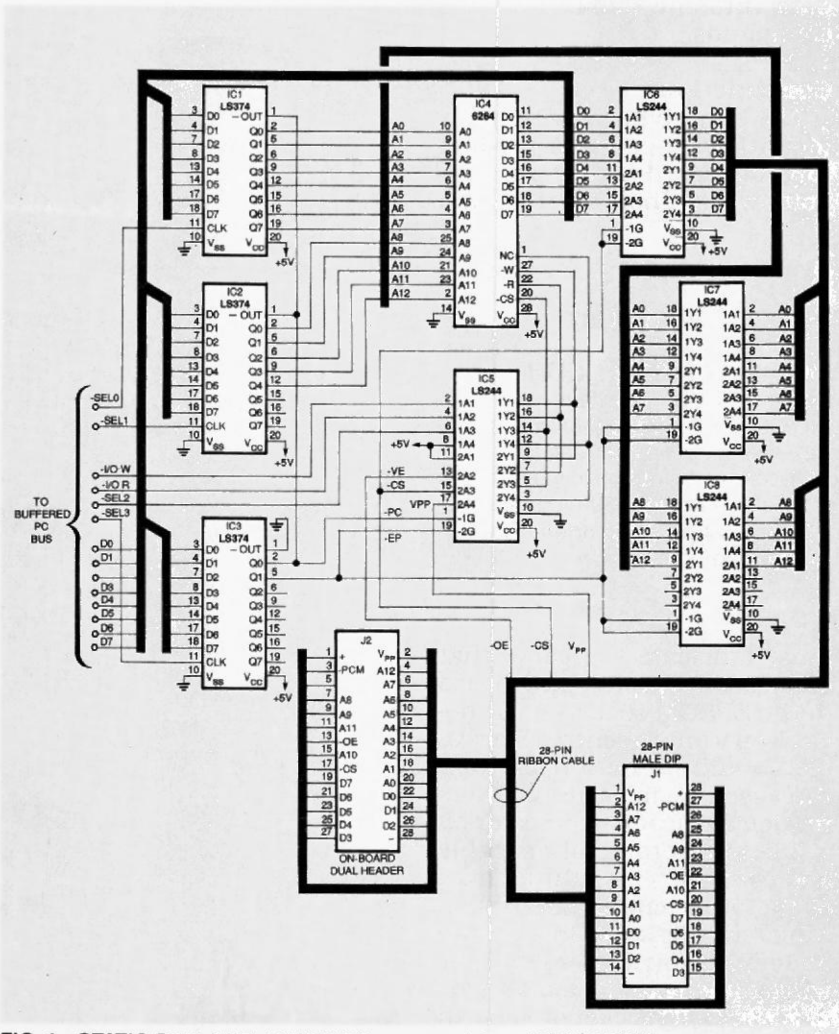

FIG. 1—STATIC RAM AND AN IBM PC can emulate a 2764 EPROM.

Circuitry

Figure 1 is the schematic of the EPROM emulator. Data from the IBM PC bus is latched on different select lines; -SEL0 is for the lower 8 bits of the address to the 6264. -SEL1 is reserved for the upper address bits, -SEL2 is to read or write data to the 6264 (reading is for verifying), and -SEL3 latches the control lines to IC3. Bit 0 from IC3 (-PC) enables the buffers for IBM PC communication and bit 1 (-EP) enables EPROM communication.

On the PC side, three 74LS374 latches (IC1–IC3) input the fully-buffered address and control lines and a 74LS244 buffer (IC5) inputs the control lines that aren't fully buffered. Three 74LS244 buffers (IC6–IC8) allow communication with the target project. A ribbon cable with a 28-pin male DIP on the end (J1) plugs into the target project's EPROM socket. The EPROM signals are accessed from a 50-pin header on the prototyping board.

Listing 1 is a Turbo Pascal program that takes a binary file that has been assembled for the target project and loads binary data into the 6264 using I/O commands.

Construction

The prototype was built by mounting all parts on the prototyping board and by point-to-point wiring them together. You must fabricate the cable that connects the prototyping board to the EPROM socket in the device you want to control. The cable should be no longer than three feet. Install a 28-pin dual-row female header connector on a 28-conductor ribbon cable and connect pins 1 through 28 of the other end of the ribbon cable (wire by wire) to a 28-pin male DIP. You can solder the wires directly to the female side of a 28-pin IC socket or you can install a 28-pin male IDC (insulation displacement connector) DIP. Note that the prototyping board contains a 50-pin male header, but only pins 1 through 28 are used. Be sure to connect the female header connector only to pins 1 through 28 of the header.

The fake EPROM is now ready to trick a circuit into thinking that a real EPROM is in place. Ω

PARTS LIST

IC1–IC3—74LS374 8-bit latch
IC4—6264 8K × 8 static RAM
IC5–IC8—74LS244 8-bit buffer
J1—28-pin male DIP (see text)
28-conductor ribbon cable (three feet, maximum), 28-pin dual-row female header connector, prototyping board.

LISTING 1

```
program feprom (input, output);
(this program will test and load the Eprom simulator 6264 8k
memory. The first parameter if specified will be the load file
and program will terminate, if no parameter is specified, then
the program will go into interactive mode. Uses Turbo Pascal V3.)

const
        port_add_low  - $300;     (6264 low byte address register)
        port_add_high - $304;     (6264 high byte address register)
        port_data     - $308;     (6264 data register)
        port_control  - $30C;     (buffer control register)
                                  (bit 0 - pc, bit 1 -eprom)store
        control_pc    - $02;      (buffer control setting, pc)
        control_eprom - $01;      (buffer control setting, eprom)
        max_mem       - $1FFF;    (max size of eprom (0 - 1FFF)  )
type
        str = string [80];
        arr = array [0..max_mem] of byte; (pc array to store file)
(set the buffers to communicate with pc)
procedure set_pc;
begin
        port [port_control] := control pc;
end;
(set the buffers to communicate with cable)
procedure set_eprom;
begin
        port [port_control] ;= control_eprom;
end;
(write proper hi and low byte address to latches)
procedure write_address(add : integer);
var
        high : integer;
        low  : integer;
begin
        low  := add and $FF
        high := (add and $FF00) div $0100;
        port[port_add_low]  := low;
        port[port_add_high] := high;
end;
(check the ram bit wise)
procedure bittest;
var 1 : integer;
    p : integer;
begin
        writeln ('Bit test');
        i := 0;
        while (i <= max_mem) and (not keypressed) do
begin
        write_address(i);
        port[port_data] := $00;
        if port[port_data] <> $00 then
          writeln('0 Error at ',i:0,' ',port[port_data] :0);
        port[port_data] := $FF;
        if port[port_data] <> $FF then
          writelin('FF Error at ',i,' ',port[port_data] :);
        i := i + 1;
end;
writeln('Bit test done');
end;
(check the ram locations)
procedure numbertest;
var i : integer;
        c : integer;
begin
```

```
 writeln('Number test');
 c := 0;
 for i := 0 to max_mem do
       begin
         write_address(i);
    port[port_data] := c;
         c := c + 1;
         if c > 255 then c := 0;
end;
writeln('Last number was ',c:0);
c := 0;
for i := 0 to max_mem do
  begin
        write_address(i);
        if port[port_data] <> c then
          writeln('Number Error at ',i:0,' ',
            port[port_data],' should be ',c:0);
        c := c + 1;
        if c > 25 then c := 0;
end;
writeln('End Number test');
end;
(check the buffers)
procedure buffertest;
begin
        writeln('Buffer test');
        write_address (0);
        port[port_data] := 55;
        set_eprom;
        if port[port_data] = 55 then
          writeln('Buffer error');
        writeln('End Buffer test');
end;
(verify that the ram has the same as the array)
procedure verfile(var a : arr);
var c : integer;
begin
        writeln('Verifying file');
        for c := 0 to max_mem do
          begin
            write_address(c);
            if port[port data] <> a[c] then
              writeln('Load file error at ',c:0,' ',
                 port[port_data] :0, should be
',a[c]:0);
     end;
end;
(load array from file)
procedure loadafile(s : str; var a : arr);
var f : file of arr;
begin
        writeln('Reading file',s:length(s) );
        assign (f,s);
        reset(f);
        read (f,a);
        close(f);
end;
(load array and transfer to ram)
procedure readfile(s : str);
var c : integer;
    a : arr;
begin
  loadafile(s,a);
  set_pc;
  writeln('Transferring file ',s : length(s) );
```

```
 for c := ) to max_mem do
       begin
         write_address(c);
         port[port_data] := a[c];
       end;
 verfile(a);
 writeln('File Load done');
 set_eprom;
end;
(get filename and transfer file to ram)
procedure loadfile;
var s : str;
begin
        writeln;
        write('Enter filename ---'); readln(s);
        readfile(s);
end;
(get filename and verify against the ram)
procedure verifyfile;
var s : str;
    a : arr;
begin
        writeln;
        write ('Enter filename ---'); readln(s);
        loadafile(s,a);
        verfile(a);
end;
(mainline)
procedure main2;
var c    : char;
    done : boolean;
begin
  while not done do
    begin
      writeln;
      writeln('A) Bit test');
      writeln('B) Number test');
      writeln('C) Buffer test');
      writeln('D) Load and verify binary file');
      writeln('E) Verify against binary file');
      writeln;
      writeln('Z) to quit');
      writeln;
           write ('Enter choice ---'); readln (c);
           set_pc;
           case c of
             'A', 'a' : bittest;
             'B', 'b' : numbertest;
             'C', 'c' : buffertest;
             'D', 'd' : loadfile;
             'E', 'e' : verifyfile;
             'Z', 'z' : done := true;
         end; (case)
     set_eprom;
         end;
end;
(See if parameter, if so then load the file and
exit.
If not then go into interactive mode)
begin
  if paramount = 1 then
    readfile(paramstr(1))
  else
    main2;
  end.
```