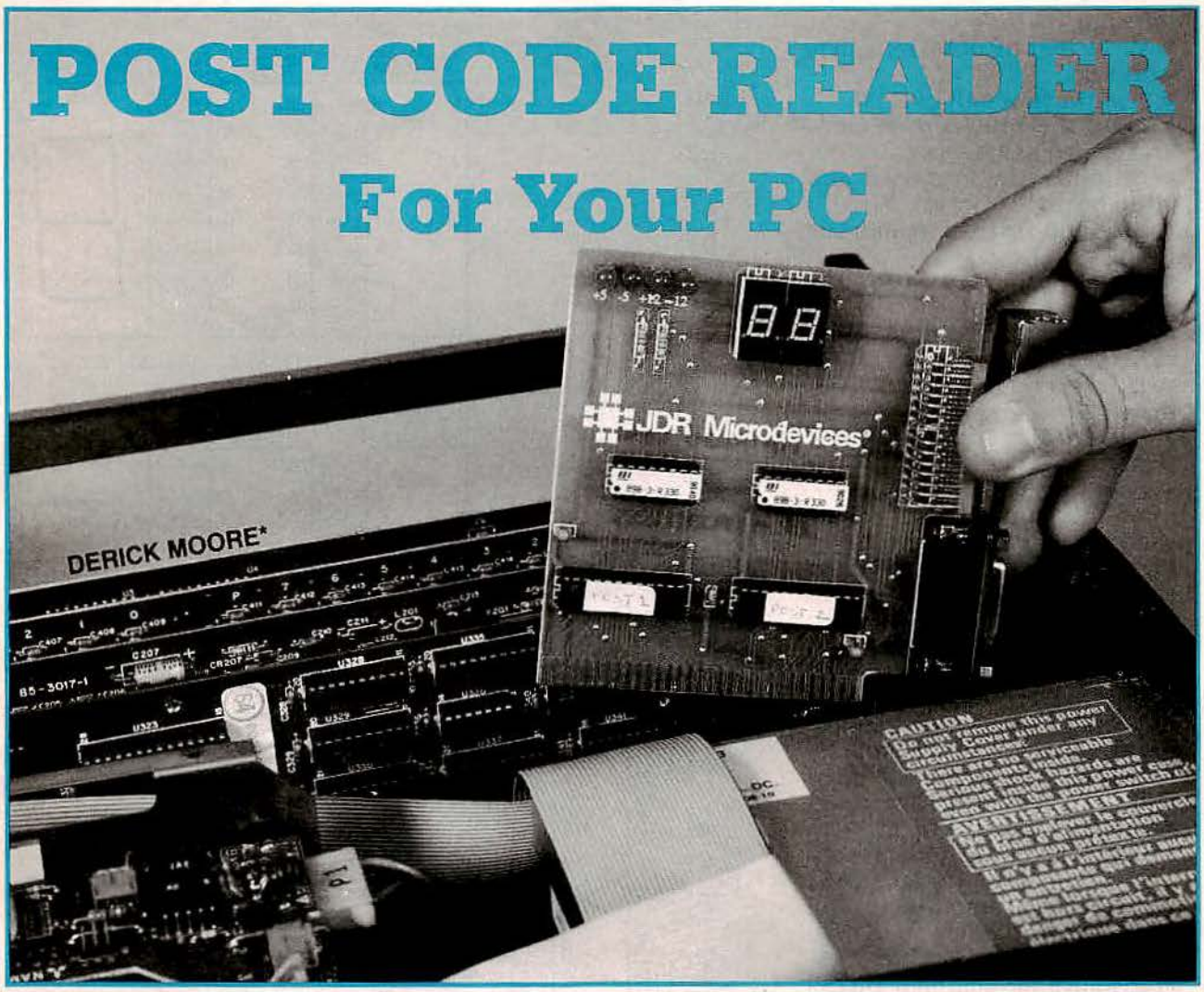
POST CODE READER
For Your PC



Your PC is trying to tell you something—and a couple of GAL's can tell you what!

LIKE THE VICTIM IN SOME CHEESY murder mystery who scrawls his dying message in the sand, your IBM-PC compatible computer might be trying to communicate with you.

Every time your PC boots, it performs a power-on self test or POST. The POST verifies all the major subsystems of your PC. If they appear live and well, the BIOS (basic input/output system) continues with the boot sequence. If any test fails, testing halts, and the value latched in I/O port 080x indicates what went wrong. (Throughout this article, an *x* appended to a number indicates that it's a hexadecimal value; decimal values have no special markings.)

The problem is that few motherboards have any way of displaying POST code values—and that's where we can help. This article describes a low-cost POST-code display board (we call it PCODE) that you can build in an hour or two. Complete kits are available, as are bare boards. (See Parts List.)

The heart of this project consists of two custom-programmed Generic Array Logic IC's (also available separately), commonly known as GAL's. Before delving in to the construction details, we'll provide some background on GAL technology and how to design with GAL's.

PC boot sequence

When a PC boots, the CPU begins executing the code at a special location in the BIOS ROM. One of the first things the BIOS does is test and initialize circuitry on the motherboard and expansion cards. Both what it does and the order it does it depend on the BIOS vendor and the version of the BIOS. However, the general sequence is:

- Check the CPU registers.
- Set up the 8253/54 timer for RAM refresh.
- Set up the DMA IC for RAM

*Director of Engineering, JDR MicroDevices

refresh.

- Verify that the RAM refresh is working.
- Test the first 64K of RAM.
- Load the interrupt vectors and assign stack space.
- Initialize the keyboard and video board.
- Size and test the remaining RAM.
- Initialize the COM, LPT, and game ports.
- Initialize the floppy disk drive(s).
- Initialize the hard disk drive(s).
- Scan and link the user ROM's, if any.
- Boot the PC.

As the computer performs each test, it updates the value in I/O port 080x. If a test fails, the CPU typically emits several beeps and then halts. The beep codes can be of some value in diagnosing problems, but typically they're neither specific nor comprehensive. So a simple device that can decode and display the value in I/O port 080x will be a very effective diagnostic tool. Ergo PCODE.

Note that different BIOS manufacturers use different code values, so you must be careful about how you interpret the values displayed by PCODE. Several values for common BIOS's appear in Table 1, but you should check with your BIOS vendor for accurate, up-to-date values.

Circuit overview

The procedure is really quite simple. We need to latch the value on the lower eight bits of the data bus whenever the CPU performs a write to I/O port 080x. Figure 1 shows a functional diagram of what we want to happen. Our goal is to achieve the $\overline{\text{SELECT}}$ signal shown in the figure.

Port 080x may be represented in binary as 00 1000 0000. In other words, the circuit needs to capture the condition when address lines A0–A6, A8, and A9 are all low, and only A7 is high. Address lines A8 and A9 must also be low, otherwise the circuit would trigger on multiples of 080x, (i.e., 0180x, 0280x, and 0380x). So note that, except

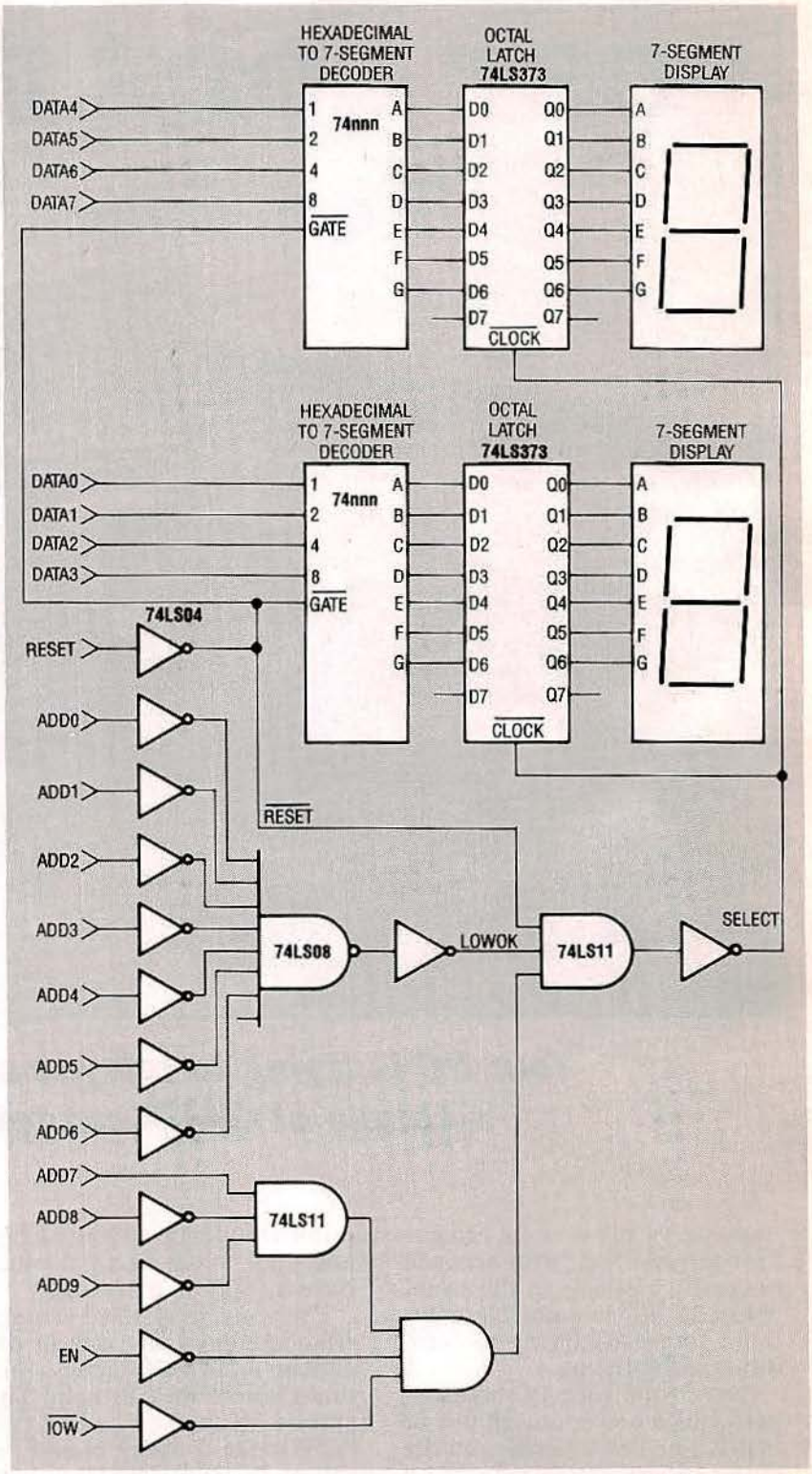


FIG. 1—FUNCTIONAL DIAGRAM provides high-level view of desired decoding. The goal is to get a Select signal that will latch data whenever the CPU writes to I/O port 080x.

for A7, we take the inverted states of A0–A9. That gives the raw port address; we also must combine the states of three control signals to achieve $\overline{\text{SELECT}}$.

Working down from the top, we combine the low states of the RESET, I/O write ($\overline{\text{IOW}}$), and memory access enable (AEN) signals. RESET is normally low; it goes high only during power-up, or when the user presses the reset switch (on a non-IBM computer). $\overline{\text{IOW}}$ is normally high; it

goes low each time the CPU writes to an I/O port. AEN is normally high; it goes low when a DMA device (e.g., a disk controller) seizes the bus.

Note in Fig. 1 that RESET also drives the $\overline{\text{GATE}}$ inputs of the two display decoders. In other words, as long as the machine is not being reset, the "gates" are open and the decoders can do their jobs. Those decoders sit on the data bus constantly decoding signals and feeding them to the display latches. The latches only *accept* new values, however, when $\overline{\text{SELECT}}$ goes low.

Although it's possible to buy IC's for some of the functions shown in Fig. 1, it's not possible for them all. The most difficult IC's to find is the pair of decoders, which have to accept hexadecimal input and deliver outputs suitable for driving a pair of dumb seven-segment LED's. In addition, combining all the inverted and true address and control signals would require numerous inverters and gates, all of which increase circuit complexity and cost, and reduce reliability.

Wouldn't it be nice if you could combine all the address-decoding logic, all the display-decoding logic, as well as the display latches and drivers, in just a few IC's? Well you can; a couple of GAL's make it easy.

PLD overview

Before we discuss the specifics of the devices used here, let's back up and look at GAL's from a more general perspective. The GAL is a member of the family of programmable logic devices (PLD's). A single PLD can replace several standard TTL devices. All PLD's share one common characteristic: they are programmable. The GAL exhibits one other characteristic that makes it extremely versatile in the design lab: Like an Electrically Erasable Programmable Read Only Memory (EEPROM), a GAL is erasable.

Figure 2 shows a simplified diagram of a portion of the 20V8 GAL used in this project. The 20V8 has a total of 24 pins. Two are used for power and ground, two for control purposes, and

TABLE 1—COMMON BEEP AND BIOS CODES

Description	IBM Beep Code	IBM AT BIOS	Phoenix 286 BIOS	AMI BIOS 2.2x	Award BIOS
CPU Register		01	01	03	07
CMOS	1 - 1 - 3	03	02	2D	0F, 1C, 1D
BIOS Checksum	1 - 1 - 4	02	03	09	
8253/4 Timer	1 - 2 - 1	04, 05	04	0F, 7E	08–0E
DMA Setup	1 - 2 - 2	06, 07	05	15, 18	08–0E
DMA Page Register	1 - 2 - 3	08	06		
RAM Refresh	1 - 3 - 1	09	08	12	
First 64K RAM		0D, 0E	09—0D, 10—1F	21, 24, 7	15
8259 #1	3 - 1 - 3	13		1B, 1E	08–0E
8259 #2	3 - 1 - 4	24, 25, 26			
0842 Keyboard Controller and Keyboard	3 - 2 - 4	0D, 2D, 35, 36, 38, 39, 3A			01–05, 2A
Address Lines 19–23 Problem		1F		60	
Video Initialization	3 - 3 - 4	22, 23		48	17, 18
Video ROM Scan in Progress	3 - 4 - 2		2D		
Video ROM Svan	3 - 4 - 3				
Real-time Clock			3B		
Serial Port			3C	93	2C
Parallel Port			3D	96	2D
Math Coprocessor			3E	90	2F
Hard Disk, Floppy Disk		3E		9C, 9F, A2	2B, 2E
Initialize Printer		3F			

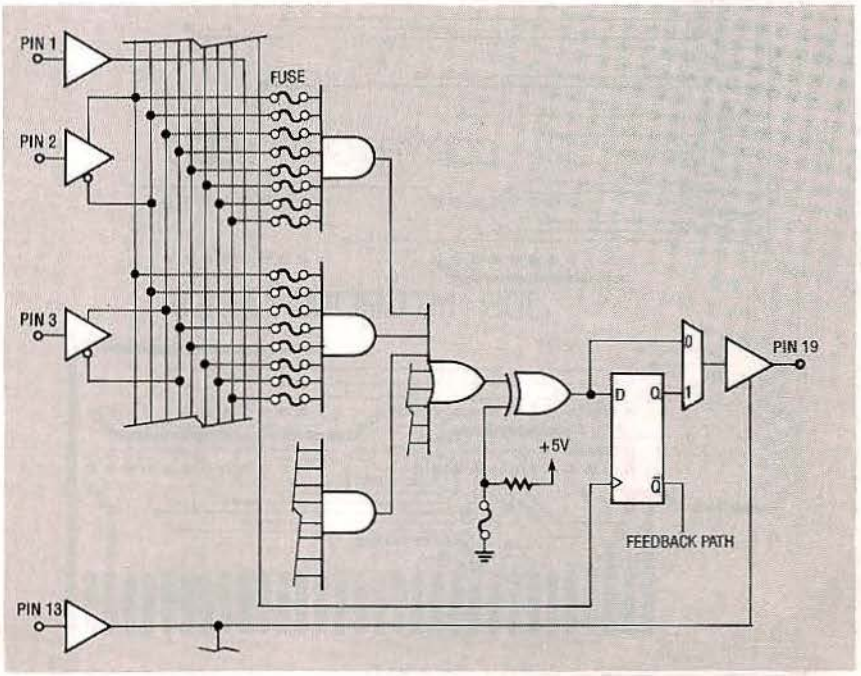
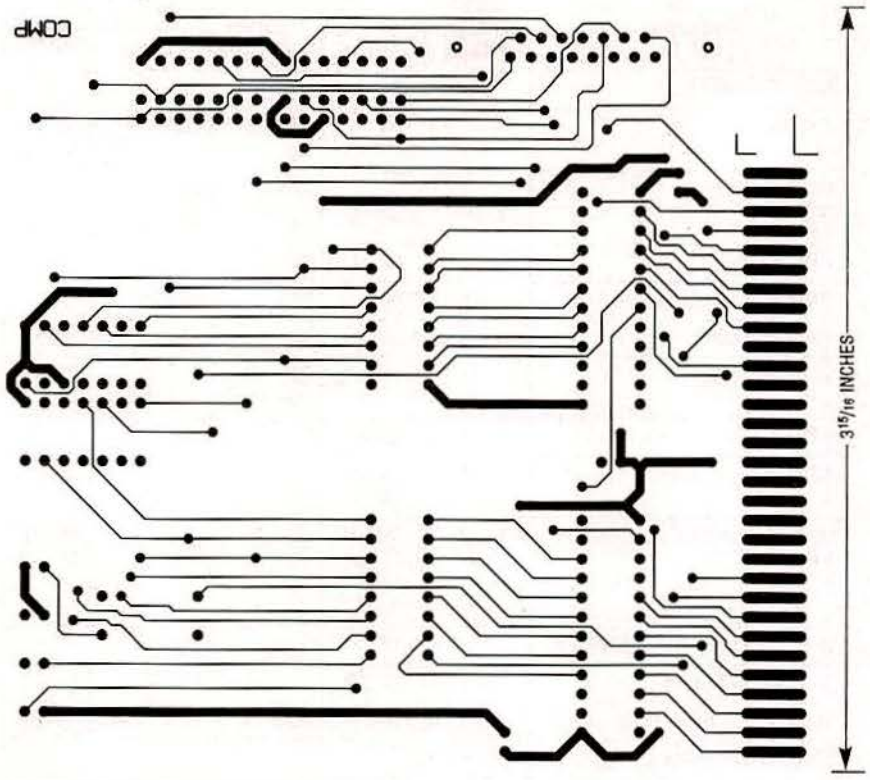
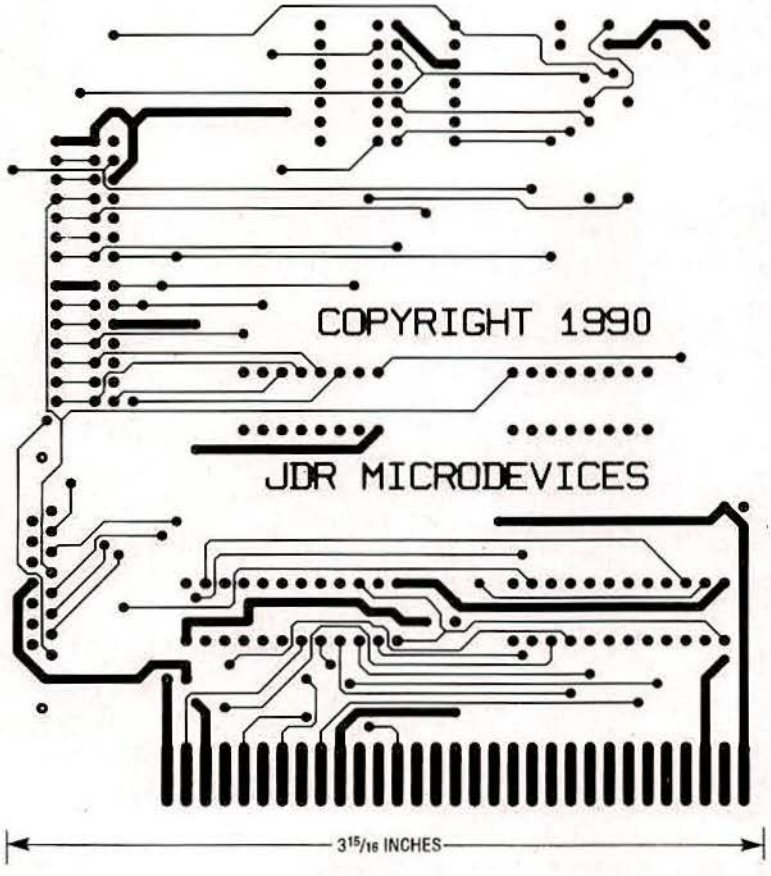


FIG. 2—SIMPLIFIED GAL VIEW indicates the logical complexity that can be hidden in a 24-pin 0.3" DIP.



PCODE COMPONENT SIDE.



PCODE SOLDER SIDE.

PARTS LIST

R1, R2—820 ohms
R3, R4—330 ohms, 8-position DIP package
C1, C2—10 µF, 16 volts, tantalum
C3—0.1 µF, 16 volts, tantalum
IC1, IC2—G20V8 GAL
DISP1, DISP2—MAN72 7-segment LED display (or equivalent)
J1—15-pin D connector, female, PC-board mount
LED1, LED4—T1-¾ green LED
LED2, LED3—T1-¾ red LED
IC Sockets, circuit board, mounting bracket and screws, etc.

Note: The following parts are available from JDR Micro-Devices, 2233 Samaritan Drive, San Jose, CA 95124, (800) 538-5000, (408) 559-1200:

- **Complete PostCode kit (including PC board, mounting bracket, all IC's, and LED's)—$39.95**
- **PC Board only—$14.95**
- **Programmed IC1 and IC2—$14.95**

CA residents add applicable sales tax.

the remaining 20 are used as general-purpose I/O. Of the 20, 12 are dedicated as inputs, and eight can be programmed as either inputs or outputs. Programming and erasing a GAL involves applying specific voltages in specific sequences. For detailed information on designing with GAL's, request data from a GAL vendor (e.g., AMD, Gould, Harris, Intel, Lattice, Motorola, National Semiconductor, Rockwell, SEEQ, Signetics, TI, Xicor, or Zilog).

Designing with GAL's

If you just want to use a GAL, you don't need to worry about the bit-banging that goes on in programming and erasing. In fact, that's the whole purpose of PLD's in general and GAL's in particular. Instead, you concentrate on the logic of what you're trying to do. Doing so allows you to be productive almost immediately.

Let's take an example. Suppose you have a conventional TTL design that you want to convert to PLD's. You might do the following:

1. Eliminate discrete inverters

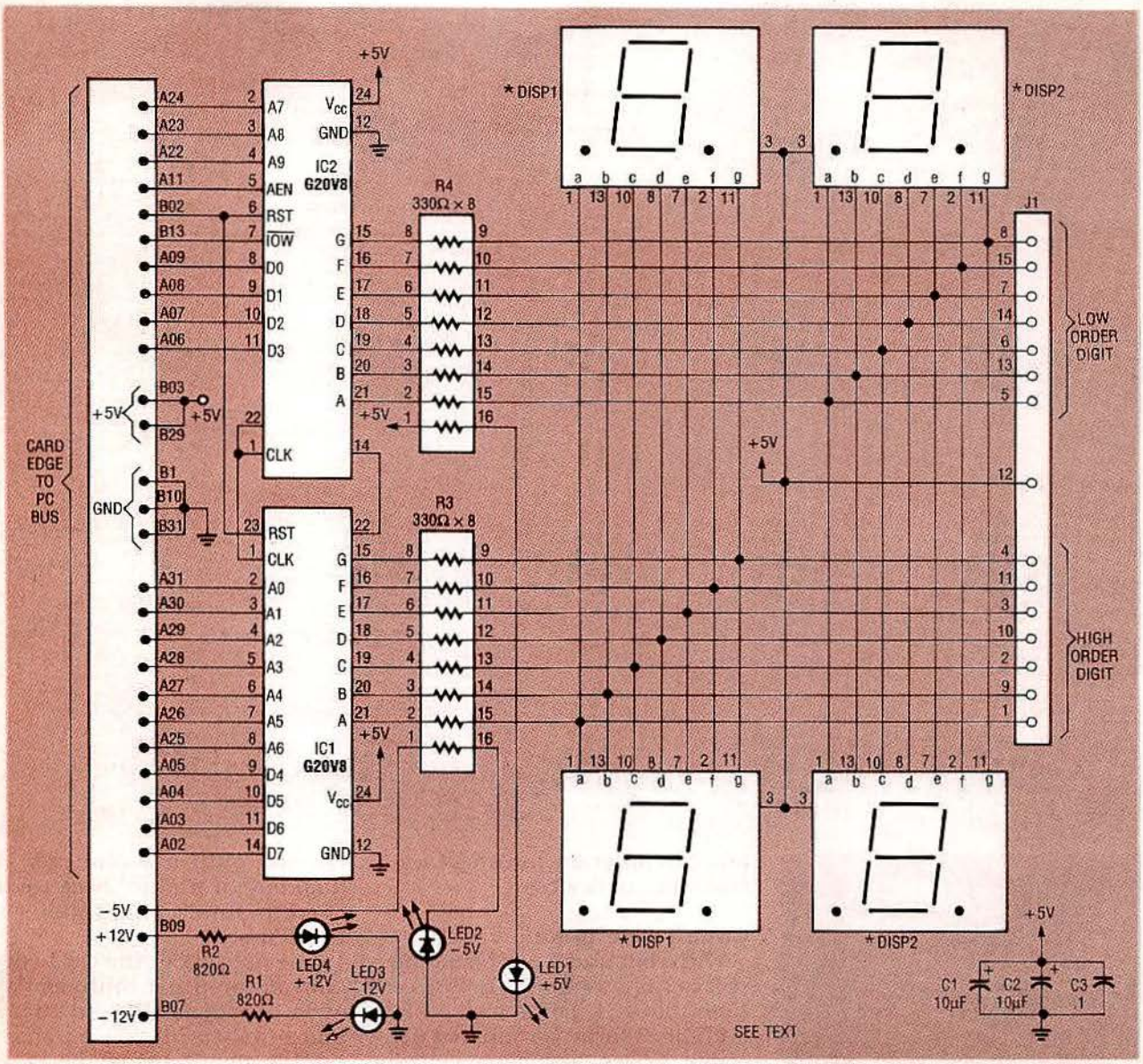


FIG. 3—COMPLETE SCHEMATIC of the POST Code display board appears here. Note that IC1 and IC2 perform all of the decoding, latching, and display driving shown in the functional diagram (Fig. 1) thereby replacing about 40 IC's.

by moving inversions to source or destination logic connections.

2. Name the signals in your TTL circuit.
3. Count the number of input and output signals.
4. If the number of inputs and outputs exceeds the capacity of the intended device, break the logic into sections for partitioning into several devices.
5. Write equations that describe the logic of your TTL design and that can be processed by a logic compiler.
6. Run the compiler and generate an object file.
7. Use the object file, a PLD programmer, and special software to "burn" the code into the device.
8. Test the device; if necessary, modify your equations, and go back to Step 6.

How would you apply that sequence to a practical problem? Take a look at Listing 1 and Listing 2, the source files for the GAL's used in PCODE. Note that the two listings share a structural similarity. Both begin with a header that specifies part name and number, date, revision number, etc. Then comes a section of comments, followed by specifications of inputs, outputs, intermediate variables, and last, the logic equations. To understand the latter, take an example logic equation from Listing 2.

```
SELECT = POST & IOW & !AEN &
LOWOK & !RESET
```

That statement says that SELECT is true when POST, IOW, and LOWOK are all true; in addition, AEN and RESET must both be false. You can see that the equation is a very compact way of notating circuit logic.

After creating the source file, run the PLD compiler. There are several commercial options including CUPL from Logical Devices. CUPL will compile the

LISTING 1—PLD SOURCE FOR IC1

```
Name     POST1;
Partno   POST1;
Date     04/07/90;
Revision 01;
Designer DERICK;
Company  JDR Microdevices;
Assembly 00001;
Location U1;
Device   G20V8;

/*************************************************************/
/*                                                           */
/*   TURNS HEX INTO 7 SEGMENT HEX DISPLAY INFO               */
/*                                                           */
/*************************************************************/
/*  Allowable Target Device Types: G20V8                     */
/*************************************************************/

/**  Inputs  **/

Pin    1    = CLOCK    ;     /* FROM SELECT ON OTHER GAL      */
Pin    2    = A0       ;     /*                +----------+   */
Pin    3    = A1       ;     /*                |A        VCC|  */
Pin    4    = A2       ;     /*                |F         B|   */
Pin    5    = A3       ;     /*                |VCC       |   */
Pin    6    = A4       ;     /*                |          |   */
Pin    7    = A5       ;     /*                |          G|   */
Pin    8    = A6       ;     /*                |          C|   */
Pin    9    = D0       ;     /*  D4            |DB        |   */
Pin    10   = D1       ;     /*  D5            |E         D|   */
Pin    11   = D2       ;     /*  D6            +----------+   */
Pin    14   = D3       ;     /*  D7                           */
Pin    23   = RESET    ;     /*                               */

/**  Outputs  **/

Pin    15   = !SEGG     ;     /*          ---A---             */
Pin    16   = !SEGF     ;     /*         |       |            */
Pin    17   = !SEGE     ;     /*         F       B            */
Pin    18   = !SEGD     ;     /*         |---G---|            */
Pin    19   = !SEGC     ;     /*         E       C            */
Pin    20   = !SEGB     ;     /*         |       |            */
Pin    21   = !SEGA     ;     /*          ---D---             */
Pin    22   = LOWOK     ;     /*                               */

/** Declarations and Intermediate Variable Definitions **/
FIELD ADDRESS = [A6..0];
FIELD DATA = [D3..0];
POST = ADDRESS:[080];

/**  Logic Equations  **/
LOWOK = POST;

SEGA.D = DATA:[0,2,3,5,6,7,8,9,A,C,E,F] & !RESET;
SEGB.D = DATA:[0,1,2,3,4,7,8,9,A,D] & !RESET;
SEGC.D = DATA:[0,1,3,4,5,6,7,8,9,A,B,D] & !RESET;
SEGD.D = DATA:[0,2,3,5,6,8,B,C,D,E] & !RESET;
SEGE.D = DATA:[0,2,6,8,A,B,C,D,E,F] & !RESET;
SEGF.D = DATA:[0,4,5,6,8,9,A,B,C,E,F] & !RESET;
SEGG.D = DATA:[2,3,4,5,6,8,9,A,B,D,E,F] & !RESET;
```

LISTING 2—PLD SOURCE FOR IC2

```
Name     POST2;
Partno   POST2;
Date     04/07/90;
Revision 01;
Designer DERICK;
Company  JDR Microdevices;
Assembly 00002;
Location U2;
Device   G20V8;

/*************************************************************/
/*                                                           */
/*   TURNS HEX INTO 7 SEGMENT HEX DISPLAY INFO               */
/*                                                           */
/*************************************************************/
/*  Allowable Target Device Types: G20V8                     */
/*************************************************************/

/**  Inputs  **/

Pin    1    = CLOCK    ;     /* WRAPPED FROM SELECT OUT PIN   */
Pin    2    = A7       ;     /*                +----------+   */
Pin    3    = A8       ;     /*                |A        VCC|  */
Pin    4    = A9       ;     /*                |F         B|   */
Pin    5    = DMAAEN   ;     /*                |VCC       |   */
Pin    6    = RESET    ;     /*                |          |   */
Pin    7    = !IOW     ;     /*                |          G|   */
Pin    8    = D0       ;     /*  D0            |          C|   */
Pin    9    = D1       ;     /*  D1            |DB        |   */
Pin    10   = D2       ;     /*  D2            |E         D|   */
Pin    11   = D3       ;     /*  D3            +----------+   */
Pin    14   = LOWOK     ;     /*                               */

/**  Outputs  **/

Pin    15   = !SEGG     ;     /*          ---A---             */
Pin    16   = !SEGF     ;     /*         |       |            */
Pin    17   = !SEGE     ;     /*         F       B            */
Pin    18   = !SEGD     ;     /*         |---G---|            */
Pin    19   = !SEGC     ;     /*         E       C            */
Pin    20   = !SEGB     ;     /*         |       |            */
Pin    21   = !SEGA     ;     /*          ---D---             */
Pin    22   = !SELECT   ;     /* WRAPS TO CLOCK ON BOTH GALS   */

/** Declarations and Intermediate Variable Definitions **/
FIELD ADDRESS = [A9..7];
FIELD DATA = [D3..0];
POST = ADDRESS:[080];

/**  Logic Equations  **/
SELECT = POST & IOW & !DMAAEN & LOWOK & !RESET;

SEGA.D = DATA:[0,2,3,5,6,7,8,9,A,C,E,F] & !RESET;
SEGB.D = DATA:[0,1,2,3,4,7,8,9,A,D] & !RESET;
SEGC.D = DATA:[0,1,3,4,5,6,7,8,9,A,B,D] & !RESET;
SEGD.D = DATA:[0,2,3,5,6,8,B,C,D,E] & !RESET;
SEGE.D = DATA:[0,2,6,8,A,B,C,D,E,F] & !RESET;
SEGF.D = DATA:[0,4,5,6,8,9,A,B,C,E,F] & !RESET;
SEGG.D = DATA:[2,3,4,5,6,8,9,A,B,D,E,F] & !RESET;
```

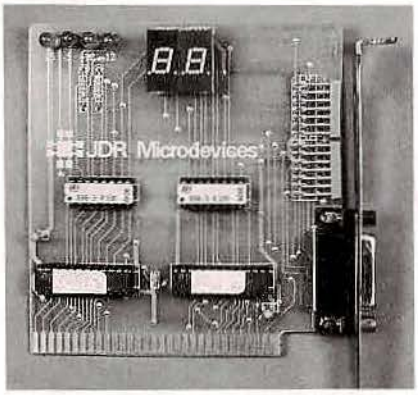


FIG. 5—NOTE THE DISPLAY SOCKETS mounted at right angles to the PC board.

source file and output an object file in several formats. For example, to create a JEDEC-compatible file for the programmer, you might enter the following DOS command line.

```
C:>CUPL -J PCODE
```

Next you would use a programmer to burn the logic specified by PCODE.JED into the selected device. PLD programming software normally reads object files in several formats, and can program and erase several types of devices.

Circuit description

With that under our belts, the rest is easy, as shown in Fig. 3. Note that IC1, DIP resistor R3, and DISP1 form a group that decodes, drives, and displays the most-significant digit; similarly, IC2, R4, and DISP2 do the least-significant digit.

Our design provides three possibilities for mounting the displays: 1) A pair of sockets mounted flat against the board. 2) Another pair mounted at a right angle so they can be seen through a hole in the mounting bracket. 3) Connections necessary for remote viewing via a 15-pin female D connector.

The first option is for "case-open" troubleshooting of a motherboard. The second is for "case-closed" motherboard monitoring. The third might be used in a situation where it's necessary to verify correct operation remotely. Regardless which display option you choose, *only use one pair of displays at a time*, because of current-sinking limitations of the GAL's.

One note about the card-edge connector pin numbers: We show the official IBM pin names and numbers; if you purchase the kit mentioned in the Parts List, the names and numbers may vary.

Construction and testing

This circuit is simple, but there are lots of interconnections, hence we recommend use of a PC board. Patterns are shown here; etched and drilled boards are also available.

Using the parts-placement diagram shown in Fig. 4, mount the two discrete resistors followed by the sockets for the IC's, resistor networks, and seven-segment LED displays. The sockets for the displays that can be seen through the mounting bracket must be mounted at a right angle, as shown in the photograph of PCODE in Fig. 5.

Mount the capacitors and dis-

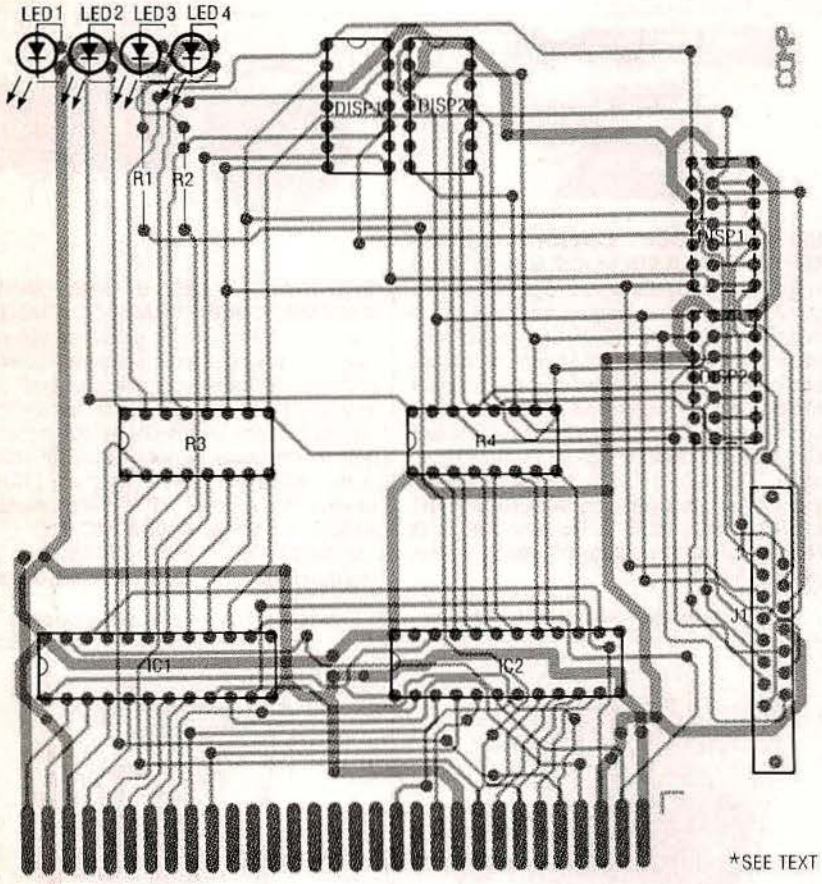


FIG. 4—MOUNT ALL COMPONENTS as shown here.

crete LED's. Last, mount J1 and attach a PC expansion-bus mounting bracket to it. Check your work carefully for shorts between traces and for bad solder joints; make any required corrections.

If you're not purchasing preprogrammed GAL's, you can burn your own using JEDEC format files available on JDR's BBS (408-559-0253). They are also available on the *Electronics Now BBS* (516-293-2283). Otherwise, you'll have to enter the logic equations (Listing 1 and Listing 2), compile them, create object files, download them to your device burner, and then burn the GAL's. We can't provide specific directions on those operations, as they depend on the software and hardware you use.

Insert the GAL's and the remaining components in the appropriate sockets on the PC board and make a final visual check. If all seems well, turn off the power and remove the case from a working PC; then insert the board in any vacant 8- or 16-bit slot. When you reapply power, the PC should boot as normal, but PCODE should display a sequence of codes as each POST test occurs. If the board seems to function properly, you can now use it to help diagnose nonfunctional units.

Other applications

PCODE was designed to test new motherboards, diagnose problems with existing boards, and provide more detailed information than the "beep" codes emitted by most BIOS's.

However, the ideas presented here have applications that extend way beyond a simple display unit. Designing with GAL's will allow you to create your own simple PC-bus interface circuits. Using a GAL to simplify the decoding, buffering, and latching of I/O ports will allow you to add your own analog and digital inputs and outputs. Now you can monitor and control anything that can be represented by a voltage or current. First POST codes, then the world! Ω