

# A PAL Programmer

*This inexpensive PAL programmer board  
fits in your IBM PC*

Robert A. Freedman

FOR THE PAST several years, I have been looking for an inexpensive PAL programmer, but I've had no luck. It seems that nothing on the market is under \$500. There are plenty of inexpensive EPROM programmers but not PAL (programmable array logic) programmers. Everyone I've talked to thinks that PAL programmers are too difficult to build at low cost or that the only market worth chasing is the multithousand-dollar universal programmer market. These universal programmers are generally too expensive for the hobbyist who has to pay for one out of his own pocket or the engineer at a large company who can't justify an expenditure of several thousand dollars for a programmer at her desk when there is one down the hall or in the next building.

Since I could not buy the kind of PAL programmer I wanted, I decided to design one myself. Many people would probably like to program a few PALs and don't want to buy an expensive universal programmer. The text box "The ZAP-A-PAL Programmer" on page 266 shows the 20- and 24-pin PAL devices that this programmer can handle.

## The PAL Programming System

ZAP-A-PAL is configured as an IBM PC adapter card (see photo 1). This eliminates

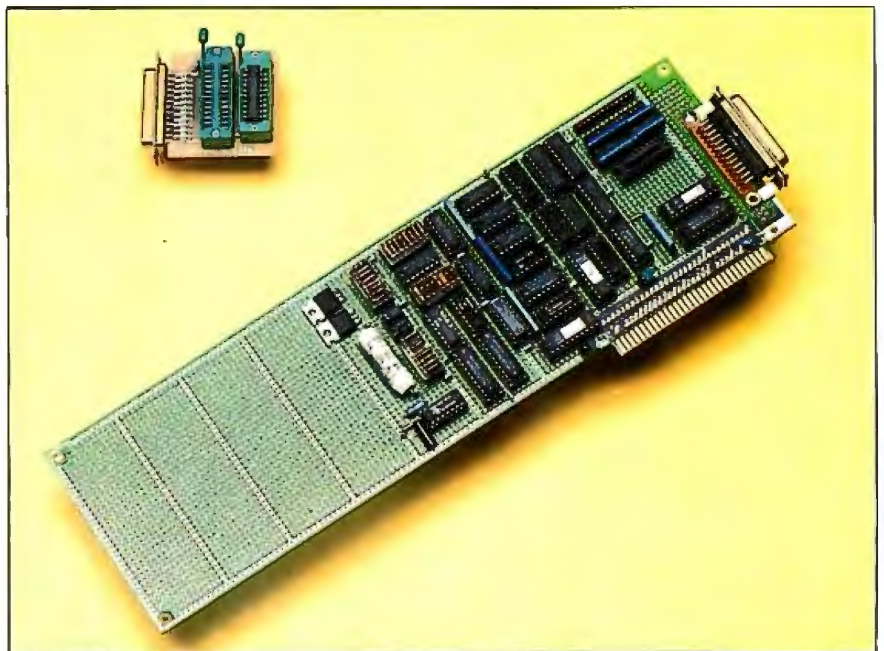
the need for a case and power supply. Also, a detachable PAL socket card with 20- and 24-pin zero-insertion-force sockets is for mounting the PALs. This card allows programming with the cabinet closed and plugs into the ZAP-A-PAL board in back of the computer during operation. Industry standard DB-25 connectors make the connection. You can use a short 25-pin shielded cable so that you can move the socket board to the front of the computer for easy access while mount-

ing and dismounting PALs.

I wrote a program, ZAPAL, whose job is to read a JEDEC file and interpret the fuse map to drive the ZAP-A-PAL card and program the PAL. [Editor's note: *The interface program fragment ZAPAL.C (source code) is available on disk, in print, and on BIX; see the insert card following page 424 for details. It is also available on BYTEnet; see page 4.*] You will need to supply the logic design com-

*continued*

Photo 1: The ZAP-A-PAL board.



Robert A. Freedman has an S.B.E.E. in computer science from MIT and works as a freelance consultant designing with microcomputers. He can be contacted at (617) 683-4659 or at P.O. Box 1348, Lawrence, MA 01842.

piler of your choice. You can use any logic compiler provided that it runs on the IBM PC and generates a JEDEC format file for output.

### Limitations

ZAP-A-PAL programs only the array fuses in bipolar PALs. At this stage of development, it does not program security fuses. ZAP-A-PAL will not program erasable CMOS programmable logic devices, and it does not program some of the PALs recently introduced by Monolithic Memories Inc. and others. ZAP-A-PAL does not program Advanced Micro Devices PALs, which use a different programming strategy than MMI, National Semiconductor, and Texas Instruments. I am working on enhancements to ZAP-A-PAL to overcome many of these limitations.

### Design Philosophy

To ensure a high degree of success for anyone attempting to duplicate this project, I set some guidelines to follow in the design of ZAP-A-PAL.

- Self-calibrating—no worry about drift or out of tolerance
- No precision resistors required
- No potentiometers
- Software entirely in C language
- No dependence on software timing loops
- Where possible, use of inexpensive, commodity components
- Open architecture, expandable to new device types
- Low cost; see the text box "The ZAP-A-PAL Programmer" on page 266.

### PAL Programming Principles

To understand the operation of ZAP-A-PAL, you must first understand how a typical PAL is organized. Refer to the device logic diagram for the 16L8 PAL in figure 4 of Vincent J. Coli's article "Introduction to Programmable Array Logic" on page 207. The axes of the array are numbered. The input lines are numbered across the top, and the product terms are numbered down the left.

In this discussion, the following terms represent

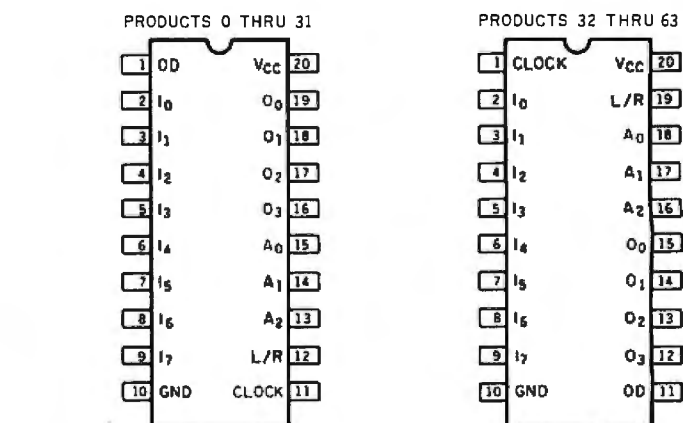
- L = "1" =  $V_{IL}$  = low = logic 0  
= GND  
H = "0" =  $V_{IH}$  = high = logic 1  
= +5 volts  
Z = "0" = resistor to +5 V (high impedance)  
HH = "2" =  $V_{IHH}$  = super-voltage  
= 11 V  
HH = "4" =  $V_{IHH}$  = program pulse  
= 11 V

$fuzno = prod\_lin * 32 + input\_lin$ . By analyzing the fuse number, you can compute all the addresses necessary to program that fuse. The input lines are organized in groups of four—that is, 0-3, 4-7, 8-11, ..., 28-31. The two low-numbered input lines in each group are connected to the noninverting and inverting inputs coming from the left of the

PAL diagram, while the two high-numbered input lines in each group are connected to the noninverting and inverting input lines coming in from the right of the diagram.

Figure 1 shows the programming pin configuration for 20-pin PALs.

The PAL is divided into two halves: Product lines 0 through 31 are in the first



### Input Line Select

Input Line Number	Pin Identification								
	I <sub>7</sub>	I <sub>6</sub>	I <sub>5</sub>	I <sub>4</sub>	I <sub>3</sub>	I <sub>2</sub>	I <sub>1</sub>	I <sub>0</sub>	L/R
0	HH	HH	HH	HH	HH	HH	HH	L	Z
1	HH	HH	HH	HH	HH	HH	HH	H	Z
2	HH	HH	HH	HH	HH	HH	HH	L	HH
3	HH	HH	HH	HH	HH	HH	HH	H	HH
4	HH	HH	HH	HH	HH	HH	L	HH	Z
5	HH	HH	HH	HH	HH	HH	H	HH	Z
6	HH	HH	HH	HH	HH	HH	L	HH	HH
7	HH	HH	HH	HH	HH	HH	H	HH	HH
8	HH	HH	HH	HH	HH	L	HH	HH	Z
9	HH	HH	HH	HH	HH	H	HH	HH	Z
10	HH	HH	HH	HH	HH	L	HH	HH	HH
11	HH	HH	HH	HH	HH	H	HH	HH	HH
12	HH	HH	HH	HH	L	HH	HH	HH	Z
13	HH	HH	HH	HH	H	HH	HH	HH	Z
14	HH	HH	HH	HH	L	HH	HH	HH	HH
15	HH	HH	HH	HH	H	HH	HH	HH	HH
16	HH	HH	HH	L	HH	HH	HH	HH	Z
17	HH	HH	HH	H	HH	HH	HH	HH	Z
18	HH	HH	HH	L	HH	HH	HH	HH	HH
19	HH	HH	HH	H	HH	HH	HH	HH	HH
20	HH	HH	L	HH	HH	HH	HH	HH	Z
21	HH	HH	H	HH	HH	HH	HH	HH	Z
22	HH	HH	L	HH	HH	HH	HH	HH	HH
23	HH	HH	H	HH	HH	HH	HH	HH	HH
24	HH	L	HH	HH	HH	HH	HH	HH	Z
25	HH	H	HH	HH	HH	HH	HH	HH	Z
26	HH	L	HH	HH	HH	HH	HH	HH	HH
27	HH	H	HH	HH	HH	HH	HH	HH	HH
28	L	HH	HH	HH	HH	HH	HH	HH	Z
29	H	HH	HH	HH	HH	HH	HH	HH	Z
30	L	HH	HH	HH	HH	HH	HH	HH	HH
31	H	HH	HH	HH	HH	HH	HH	HH	HH

Figure 1: Voltage configuration to program a 20-pin PAL. Adapted from The PAL

## PAL PROGRAMMER

half, and product lines 32 through 63 are in the second half. Depending on the half you are trying to program, the meaning of the OD and CLOCK pins reverses: OD is on pin 1 for the first half of the array, and CLOCK is on pin 1 for the last half of the array. The positions of the O<sub>3</sub>, L/R, A<sub>0</sub>, A<sub>1</sub>, and A<sub>2</sub> pins also change.

If the fuse number is less than halfway

through the array, the fuse is in the first half of the PAL and vice versa. For example, the 16L8 has 2048 fuses and 64 product terms. So if (fuzno < 2048/2) then the fuse is in the first half of the PAL; otherwise it's in the second half.

You select the input line group by placing a logic value of Z on the appropriate

*continued*

## Voltage Legend

L = Low-level input voltage, V<sub>IL</sub>

H = High-level input voltage, V<sub>IH</sub>

HH = High-level program voltage, V<sub>HH</sub>

Z = High impedance (e.g., 10 kΩ to 5.0 V)

## Product Line Select

Product Line Number	Pin Identification						
	O <sub>3</sub>	O <sub>2</sub>	O <sub>1</sub>	O <sub>0</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>
0, 32	Z	Z	Z	HH	Z	Z	Z
1, 33	Z	Z	Z	HH	Z	Z	HH
2, 34	Z	Z	Z	HH	Z	HH	Z
3, 35	Z	Z	Z	HH	Z	HH	HH
4, 36	Z	Z	Z	HH	HH	Z	Z
5, 37	Z	Z	Z	HH	HH	Z	HH
6, 38	Z	Z	Z	HH	HH	HH	Z
7, 39	Z	Z	Z	HH	HH	HH	HH
8, 40	Z	Z	HH	Z	Z	Z	Z
9, 41	Z	Z	HH	Z	Z	Z	HH
10, 42	Z	Z	HH	Z	Z	HH	Z
11, 43	Z	Z	HH	Z	Z	HH	HH
12, 44	Z	Z	HH	Z	HH	Z	Z
13, 45	Z	Z	HH	Z	HH	Z	HH
14, 46	Z	Z	HH	Z	HH	HH	Z
15, 47	Z	Z	HH	Z	HH	HH	HH
16, 48	Z	HH	Z	Z	Z	Z	Z
17, 49	Z	HH	Z	Z	Z	Z	HH
18, 50	Z	HH	Z	Z	Z	HH	Z
19, 51	Z	HH	Z	Z	Z	HH	HH
20, 52	Z	HH	Z	Z	HH	Z	Z
21, 53	Z	HH	Z	Z	HH	Z	HH
22, 54	Z	HH	Z	Z	HH	HH	Z
23, 55	Z	HH	Z	Z	HH	HH	HH
24, 56	HH	Z	Z	Z	Z	Z	Z
25, 57	HH	Z	Z	Z	Z	Z	HH
26, 58	HH	Z	Z	Z	Z	HH	Z
27, 59	HH	Z	Z	Z	Z	HH	HH
28, 60	HH	Z	Z	Z	HH	Z	Z
29, 61	HH	Z	Z	Z	HH	Z	HH
30, 62	HH	Z	Z	Z	HH	HH	Z
31, 63	HH	Z	Z	Z	HH	HH	HH

Handbook, 3rd ed., by Monolithic Memories Inc.

# Z=NIX

TECHNOLOGY EVERYONE CAN AFFORD!

## AT-286 SYSTEM

- 80286 Processor
- 512K RAM on mother board
- Clock/Calendar with battery backup
- Color Graphics Printer Card or Monochrome Graphics Printer Card
- Floppy Disk Controller/ Hard Disk Controller
- 1.2 MB Floppy Disk Drive
- AT Compatible Keyboard
- MS DOS 3.1 included

\$1395



## TURBO BAREBONE

- Four-layer Turbo Board with 256K RAM on board
- Four-layer Monochrome Graphics Card with Printer Port
- Four-layer Floppy Disk Controller Card
- One TEAC floppy drive
- 135 Watt Power Supply
- One year warranty (Storage Peripherals not included)
- MS DOS 3.1 optional \$49

\$539



## NAKAI MOUSE

- Input speed higher than any devices
- No traditional gridded pad required
- Without using power supply adaptor
- No operating space limit
- Mouse System Mouse Compatible
- Software optional

\$69



## 4 LAYER BOARD PERIPHERAL

- Mother Board w/256K \$179
- Turbo Board w/256K \$189
- Color Graphic Card \$76
- Multi I/O Card \$103
- Monochrome Graphic Printer Card \$84
- Disk Controller Card \$39

## 2LAYER BOARD

- Mother Board OK Memory \$89
- Turbo Board OK Memory \$109
- Color Graphic Card \$53
- Multi I/O Card w/Floppy Controller \$69
- Monochrome Graphic Card \$59
- Disk Controller Card \$28
- EGA Card \$229
- Basic System (Case, KB-5160 KB, 150WT P/S) \$143
- TEAC Drive \$95
- Fujitsu Drive \$90
- ASSEMBLY & TESTING FEE \$35

## NAKAI COMPANY, INC.

10527 Humbolt Street,  
Los Alamitos, CA 90720

Manufacturer Representative Wanted  
Information

**(213) 493-2516**

To Order Toll Free

**1-800-331-6083**

$I_r$  pin where ( $I_0 \leq I_r \leq I_7$ ). A logic value of Z on the L/R pin selects the low-numbered inputs in a group, while a value of HH on this pin selects the high-numbered inputs. Therefore, you can compute L/R as: ( $LR = fuzno \& 2 ? Z : HH$ ). The input signal polarity is determined by the variable `input__lin`. If `input__lin` is even, the input is noninverting, and if `input__lin` is odd, the input is inverted. The product terms are grouped eight to an output. To find the output pin ( $O_x$ ) that a fuse is on, compute  $O_x = fuzno / (32 * 8)$ . To find

the address ( $A_0, A_1$ , or  $A_2$ ) of the product term of that output, compute  $addr = fuzno \% 8$ , or modulo 8. Each 0 bit of the 3-bit address is set to Z. Each 1 bit of the address is set to HH.

### Circuit Description

In this discussion, pin numbers P1 through P24 refer to both the 20-pin and the 24-pin sockets. The actual 20-pin socket pins are mapped onto the 24-pin socket's pins. Figure 2 shows the details of the socket-board schematic.

What follows is a description of how I translated the PAL programming principles just reviewed into a board that can generate the voltages and signals required to program these devices. This will start with the pin-driver circuitry that is responsible for presenting these voltages to the socket board and its power supply. What will be described next is what's required to read the PAL to verify that the proper fuses have blown, followed by a description of the IBM PC to ZAP-A-

*continued*

## The ZAP-A-PAL Programmer

The ZAP-A-PAL can program both 20-pin and 24-pin PALs. It plugs into an IBM PC and uses commercially available logic design software. The total cost of building ZAP-A-PAL is less than \$200.

The types of PALs that ZAP-A-PAL will program are listed below:

20-pin	10L8	10H8	12L6	12H6	14L4	14H4
	16L2	16H2	16C1	16A4	16X4	
	16R4	16R6	16R8	16L8		
	16R4BP	16R6BP	16R8BP	16L8BP		
24-pin	12L10	14L8	16L6	18L4	20L2	
	20C1	20L10	20X10	20X8	20X4	
	20R4	20R6	20R8	20L8		

It will do MMI standard PALs with A, B, and D speed suffixes and -2 and -4 power suffixes for available types. It will do National Semiconductor and Texas Instruments PALs from the above list.

The following is a list of parts needed to construct the ZAP-A-PAL board. Prices may vary from those given.

Printed Circuit Board, WW	1	PAL 16R8	1	8-pin IC Sockets	1
JDR MicroDevices		DAC-08 EP	2	14-pin IC Sockets	8
Socket Module PC Board	1	LM-317 T-220, Adjust. Reg.	2	16-pin IC Sockets	16
24-pin ZIF Socket	1	LM-324, Quad Op-amp	1	18-pin IC Sockets	3
3M-Textool		LM-336, 2.5-V Reference	1	20-pin IC Sockets	7
20-pin ZIF Socket	1	LM-339, Quad Comparator	3	24-pin IC Sockets	1
RS-232C D-Sub 25-S Rt. Ang.	1	TL-497ANC	1	DALE 1HA-203 100 $\mu$ H	1
R. S. Cat #276-1521		1N4001 Diode	6	or any 100-250 $\mu$ H @ 1 amp	
RS-232C D-Sub 25-P	1	1N4740A, 10-V Zener	1	100-pF Mica Cap	1
UNC5810A Sprague	3	1N4935 Fast Recov. Diode	1	0.01- $\mu$ F Monolithic Caps	2
UNC5821A	4	100-ohm $\frac{1}{4}$ -watt 5 percent Res.	1	0.1- $\mu$ F Monolithic Caps	30
UNC5895A	1	240-ohm $\frac{1}{4}$ -watt 5 percent Res.	2	put one cap on each IC power pin	
IRFD-9123 HEXDIP Power FET	1	1.0-kohm $\frac{1}{4}$ -watt 5 percent Res.	1	15- $\mu$ F @ 20-V Tantalum Cap	4
7406	1	1.2-kohm $\frac{1}{4}$ -watt 5 percent Res.	2	22- $\mu$ F @ 25-V Tantalum Cap	2
LS138	1	2.0-kohm $\frac{1}{4}$ -watt 5 percent Res.	1	100- $\mu$ F @ 16-V Aluminum Cap	1
LS245	1	2.2-kohm $\frac{1}{4}$ -watt 5 percent Res.	1	1.0-ohm 1-watt Resistor	1
LS251	2	2.7-kohm $\frac{1}{4}$ -watt 5 percent Res.	1	Resistor SIP 1.0K x 7	1
LS259	1	5.11-kohm $\frac{1}{4}$ -watt 5 percent Res.	8	Resistor SIP 2.2K x 9	3
LS273	2	(1 percent better, but 5 percent okay)		Resistor SIP 4.7K x 7	2
LS390	1	5.6-kohm $\frac{1}{4}$ -watt 5 percent Res.	1	Resistor SIP 4.7K x 5	2
PAL 16L8	2	15-kohm $\frac{1}{4}$ -watt 5 percent Res.	1		

**MAC INKER™**

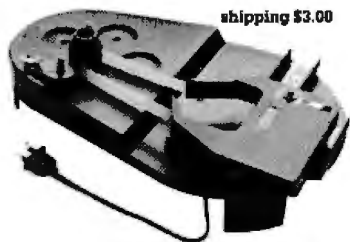
Re-ink Any Fabric Ribbon  
Automatically for less  
than 5 cents.

Dedicated Units Start at **\$54.95**

Universal Cartridge  
MAC INKER **\$68.50**

Universal Spool  
MAC INKER **\$66.95**

shipping \$3.00



Lubricated, Dot Matrix Ink \$3.00,  
bottle available in black/brown/  
red/green/yellow/purple/  
orange/gold and silver.

Over 50,000 MAC INKER(s) in the field.

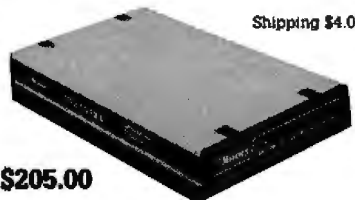
Over 7500 printers supported.

**MERCURY MODEM**

Really 100% Hayes\* Compatible.

- 300/1200 baud.
- audio monitor/front panel lights.
- 18 months warranty.

Shipping \$4.00

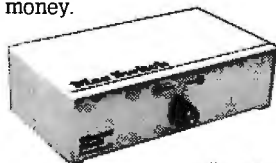


**\$205.00**

\*Hayes is a trademark of Hayes Microproducts.

**DATA SWITCHES**

All types, all lines switched, all metal,  
heavy duty switch, elegant design, best  
value for money.



2 Ports Parallel or Serial **\$ 75<sup>00</sup>**

4 Ports Parallel or Serial **\$150<sup>00</sup>**

2 Computers/2 Printers  
Parallel or Serial **\$150<sup>00</sup>**

We have cables too. Please inquire or  
specify at time of order.

Order Toll Free.

Call or write for free brochure.

**1-800-547-3303**

In Oregon 503-626-2291 (24 hours line)

**Computer  
Friends®**

6415 SW Canyon Ct., #10, Portland, OR 97221,  
telex 4949559

Dealer inquiries welcome.

**PAL PROGRAMMER**

## *ZAP-A-PAL uses serial-input BiMOS latched drivers as pin drivers.*

PAL interface circuitry and calibration  
procedures.

**Pin Drivers**

Sprague has a series of chips called serial-input BiMOS latched drivers that I use as pin drivers in ZAP-A-PAL. The two types are source drivers and sink drivers. Both have  $n$ -bit shift registers,  $n$ -bit latches, and high-current, high-voltage, Darlington output transistors. I constructed a shift register 28 bits long using three strings of these shift registers, making it possible to drive each PAL pin either to ground, to  $V_{IH}$ , or to Z. Resistor SIPs (single in-line packages) are used to apply  $V_{IH}$  to all socket pins. These establish the logic high level Z on any pins not overridden by one of the pin-driver outputs being asserted and enabled. Each string is controlled by data bits 0, 1, and 2 written into the I/O port of ZAP-A-PAL and presented on lines SHD0, SHD1, and SHD2 as il-

lustrated in figure 3.

The first string consists of four UCN5821A 8-bit sink driver chips and is fed by data bit 0. This string pulls the PAL socket pins down to near ground. You can disable the second chip in the chain with the signal ENCL via software control. This floats either pin P1 or pin P13—either of which can be a CLOCK input to the PAL—to allow reading of the state of the selected fuses.

The second string consists of three chips: a UCN5810A 10-bit source driver, a UCN5895A 8-bit source driver, and another UCN5810A. This string is controlled by data bit 1 and applies the voltage  $V_{IH}$  to the PAL being programmed. The first chip drives L/R,  $A_0$ ,  $A_1$ , and  $A_2$  to  $V_{IH}$ , as required on the 10 PAL output pins. The UCN5895A pulls up the OD signal (either pin P1 or P13, depending on the half of the PAL you're writing to) to  $V_{IH}$ . This chip's outputs are enabled by the signal ENCH, which is also under software control. The last chip in this string applies  $V_{IH}$  to any of the PAL socket's 10 input pins (P2 through P11).

The third string consists of a single UCN5810A. Fed by data bit 2, this chip applies programming pulses to any of the PAL socket's 10 output pins (P14 through P23). Since this chip is parallel with the

*continued*

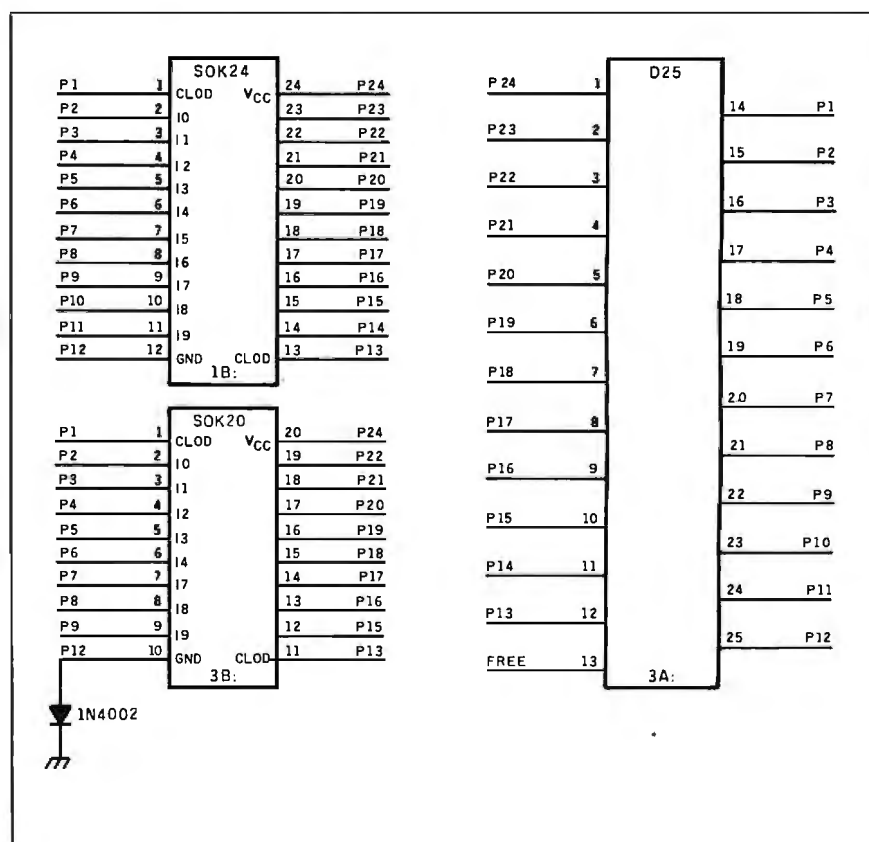


Figure 2: Schematic for the socket board.



## IDETIX™

### DIGITAL CAMERA MOS Imaging System

#### SYSTEM INCLUDES:

MOS sensor based, IBM PC compatible camera with C-Mount lens. High speed controller board. Demonstration software and subroutine library.

Adjustable frame size and resolution to 1024 × 512 pixels using the MOS digital image sensors, IS32A and IS256 OpticRAM™.

Rugged and reliable camera head for industrial environments. The high speed differential driver/receiver camera head measures 4.3"L × 1.5"H × 3"W.

#### MACHINE VISION APPLICATIONS INCLUDE:

FACTORY INSPECTION  
ROBOTICS  
REPROGRAPHICS

PROCESS CONTROL  
MANUFACTURING AUTOMATION  
SIGNATURE VERIFICATION

#### UNDER \$500 IN OEM QUANTITIES.

Micron Technology, Inc., a world leader in semiconductor development and manufacturing, presents the IDETIX™ Imaging System, a low cost alternative for machine vision applications.

For more  
information contact:

**MICRON**  
TECHNOLOGY, INC.

Systems Group  
2805 E. Columbia Road  
Boise, Idaho 83706  
(208) 386-3800

*You can disable the  
Darlington outputs  
within each chip  
without affecting  
the contents  
of the latches.*

first chip of the second string, you must take care that the two chips act only on mutually exclusive pins. This chip's output enable pin is controlled directly from a timing PAL (PAL-2) to be described later.

By sending the pin configuration data serially, you can build in protection against driving a pin high and low at the same time. You accomplish this by passing the data stream through a PAL (PAL-3) that ensures that no more than one driver is active for each bit. You can disable the Darlington outputs within each chip without affecting the contents of the latches. This allows precise timing control of the application of voltage pulses to the pins of the PAL being programmed.

Since only one driver at a time can be on for a given pin, the data sent to the shift register can take on only a limited number of values. See figure 4 for these values and how they relate to the shift register and PAL pins.

You load the shift register by doing an output instruction to I/O address 102 with the desired value (0, 1, 2, 4) in AL. For example, in C this would be `outportb(0x102,1)`. You must load the shift register with 28 values, although only 20 or 24 of these values are actually used. This is because the position of each value in the shift register determines the output on a designated PAL pin, and this register can be loaded only serially. See table 1 for several examples of the shift register's contents for a programming operation. Once all 28 shift-register positions are loaded, you must strobe them into the latches by writing a 1 followed by a 0 to I/O address 108, toggling the STR line. [Editor's note: All addresses are in hexadecimal.]

#### Power Supply

The 12 V from the IBM PC's power supply is not quite high enough to provide the 11.75 V needed for MMI's PALs because of the voltage drops in the drivers, so a booster circuit was designed (see figure 5). This is a switching regulator using the TL-497 chip. It operates by momentarily

*continued*

UNIX is a trademark  
of AT & T Bell Laboratories.

**M680UX  
Series**



## A HARD ACT TO FOLLOW

**SORD's M680UX Series  
sets a new standard in performance.**

- Operable with UNIX™
  - Astonishing speed: use of 32-bit Motorola 68020 CPU
  - Easy expansion: VME bus architecture.
- Also available on OEM basis.

**SORD®**

SORD COMPUTER CORPORATION  
Kyobashi K-1 Bldg., 2-7-12 Yaesu, Chuo-ku, Tokyo 104 Japan  
(03) 281-8130 New York: (212) 759-0140  
London: (01) 631-0787 W. Germany: (02161) 66-3077

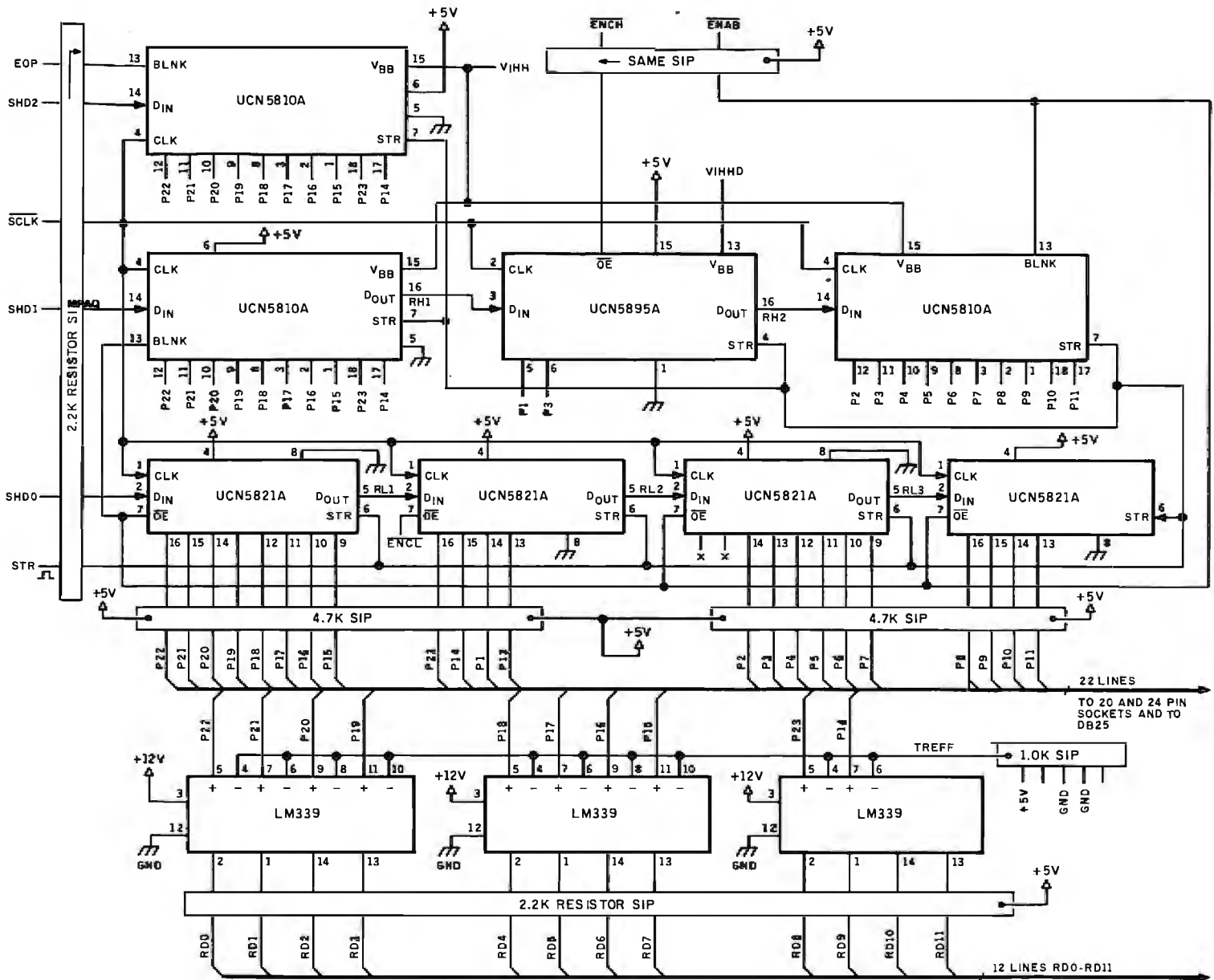


Figure 3: Schematic for the pin drivers and verify circuits.

shorting an inductor between 12 V and ground. When the coil is disconnected from ground, the energy is dumped through a diode into an output capacitor, raising its voltage with each pulse. The switching regulator chip monitors the voltage on the capacitor and, when it is high enough, the pulsing stops.

A resistive divider sets the output voltage to approximately 15 V. This is regulated down to the desired voltages by linear regulators driven by digital-to-analog converters. Thus, software can define the various voltages needed for different PAL types. The booster output is software-selectable from 15 V through

23 V, but currently only 16 V is necessary.

### Logic Verify Circuitry

During the verify portion of the ZAPAL programming procedure, you must read the logic level presented on the output pin after it has been pulsed to see if the fuse was blown. This presents a problem, as you must read a TTL level from a pin that a moment ago had a 12-V high-current pulse on it. What is needed is a device that can withstand the programming pulse and live to discriminate a TTL level to some degree of precision.

The LM-339 quad comparator is inex-

pensive and common. Three LM-339s read the 10 possible output pins, leaving two comparators free for other use. The reference inputs of the 10 comparators are tied to a reference made out of a resistor SIP. A SIP is preferred because the ratio of the resistors is more important than their absolute values for determining the reference voltage. Since SIP resistors are manufactured together, their resistive values are closely matched. The open collector outputs of the LM-339s are pulled up by other resistor SIPs to +5 V and are connected to the inputs of a pair of 74LS251 octal multiplexers. These are run to bit 0 of the data bus and respond to I/O read commands to the 16 consecutive locations at hexadecimal 100 through 10F.

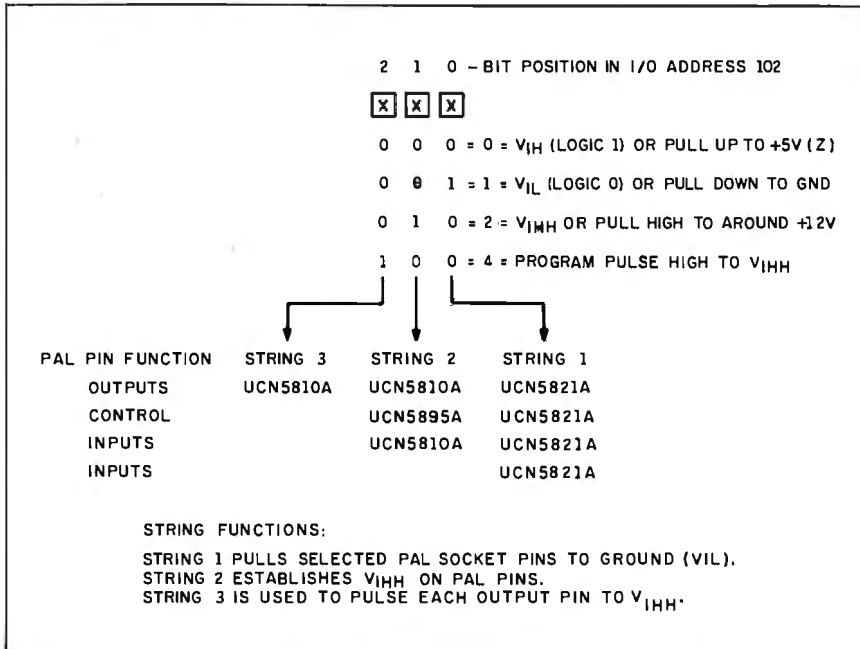
### IBM PC Interface

A PAL 16L8 (PAL-1) decodes address lines SA9 through SA3, IOW, IOR, and AEN on the IBM bus to produce four strobes for an 8-bit programmable latch (LS259), a 1-of-8 decoder/demultiplexer (LS138), and two eight-input multiplexers (LS251). Referring to figure 6, you can see that the LS251s read back individual signals to the IBM PC on bit 0 of the data bus as described above. The LS138 further decodes three strobes: two for the LS273 DAC latches and one (SCLK) that clocks data into the pin-driver shift registers. The shift data is sent to the BiMOS shift registers, one pin for each assertion of SCLK. The shift data is sent via bits 0, 1, and 2 of the data bus.

A 74LS259 is used as a set of programmable latches. You can program each bit to stay high or low until next accessed. These configure the ZAP-A-PAL board by software and actuate various parts of the circuit. All latches in the 74LS259 come up cleared on a computer reset. I/O address 10A enables the outputs of the UCN5895A. It puts  $V_{IH}$  on the OD pin of the PAL to set it up for programming. Address 10B enables the UCN5821A sink driver to pull the CLOCK pin on the PAL low and to pulse it to  $V_{IH}$  momentarily to clock the data onto the output pins for the verify operation. When address 10E is 0, it inhibits operation of the booster switching regulator, thus reducing power consumption. Address 10D can select the booster output voltage to one of two levels: 0 provides about 15 V, and 1 gives about 24 V. Address 10F controls the level of the TRIG signal, which initiates a program cycle when toggled. Check table 2 for a summary of the ZAP-A-PAL board's I/O addresses and an explanation of their functions.

One critical aspect of this project is the duty-cycle requirements in the PAL programming specification. Basically, the

*continued*

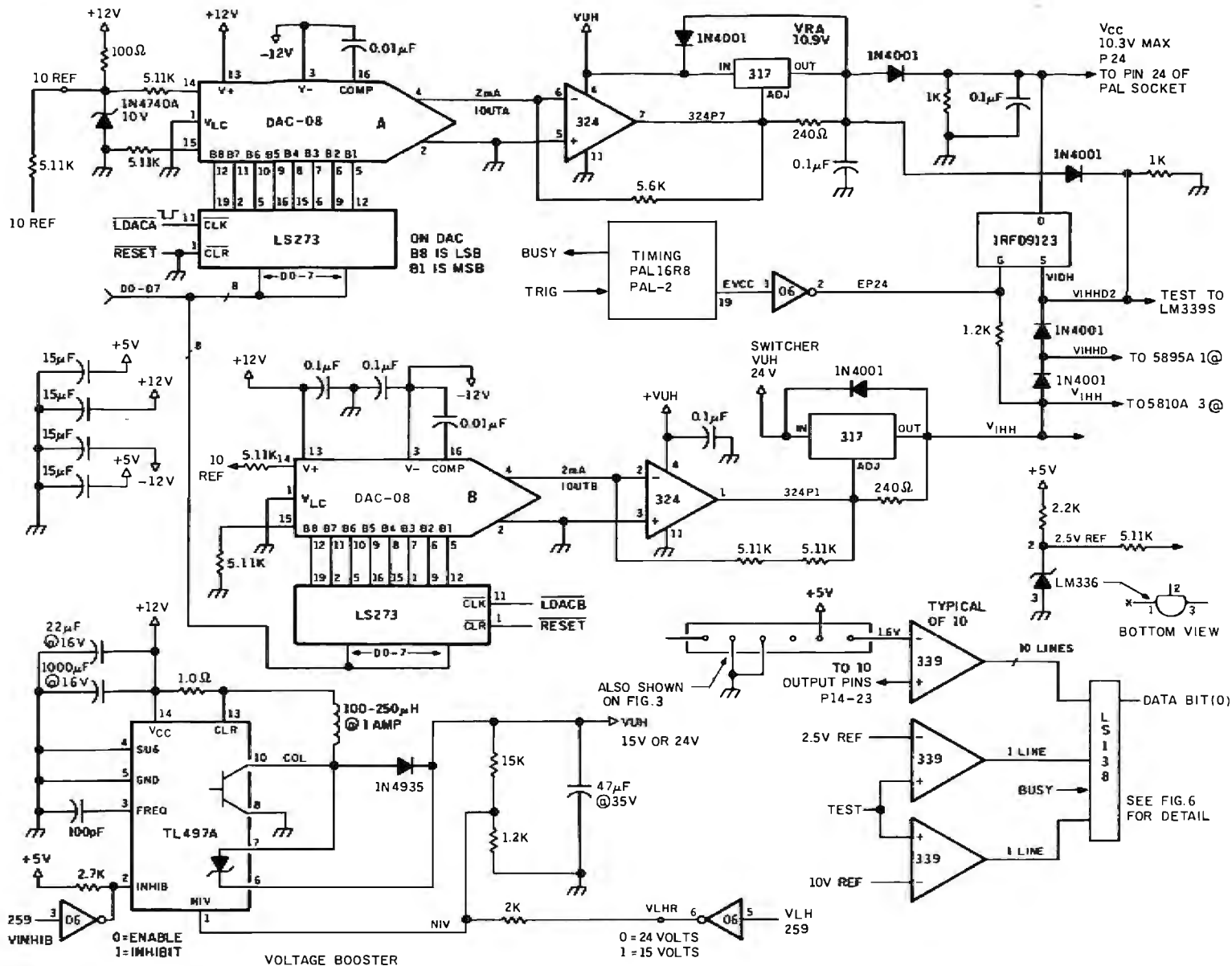


**Figure 4:** Relationship between a programming value written into ZAP-A-PAL's I/O port and the pin drivers.

**Table 1:** Examples of data values for the pin drivers. This is how the data is loaded into the shift register. Since the shift register is loaded serially, the unused portions of the register (designated by the lack of a pin number) must be padded with zeros to position the driver values at the correct PAL pin. Note that the pin numbers refer to the pin numbers on the 24-pin socket.

	Outputs	Control	Inputs
Pin	2 2 2 1 1 1 1 1 2 1	1	1 1
number	2 1 0 9 8 7 6 5 3 4	1 3 - - - - -	2 3 4 5 6 7 8 9 0 1
	0,0,0,0,0,0,0,0,0,0	0,0,0,0,0,0,0,0	0,0,0,0,0,0,0,0,0,0
	0,0,0,0,0,0,0,0,0,0	2,1,0,0,0,0,0,0	0,0,0,0,0,0,0,0,0,0
	0,0,0,0,0,0,0,0,0,0	1,2,0,0,0,0,0,0	0,0,0,0,0,0,0,0,0,0
20-pin	4,0,0,0,0,0,0,0,2,0,0	2,1,0,0,0,0,0,0	1,2,2,2,2,2,2,2,0,0
20-pin	0,0,0,0,0,0,0,0,4,0,0	1,2,0,0,0,0,0,0	2,2,2,1,2,2,2,2,0,0
24-pin	2,2,2,0,0,0,0,0,2,4	1,2,0,0,0,0,0,0	1,2,2,2,2,2,2,2,0,0
			Fuse #2
			Fuse #1100
			Fuse #3200

**Figure 5:** Schematic of the programmable voltage generator circuits.



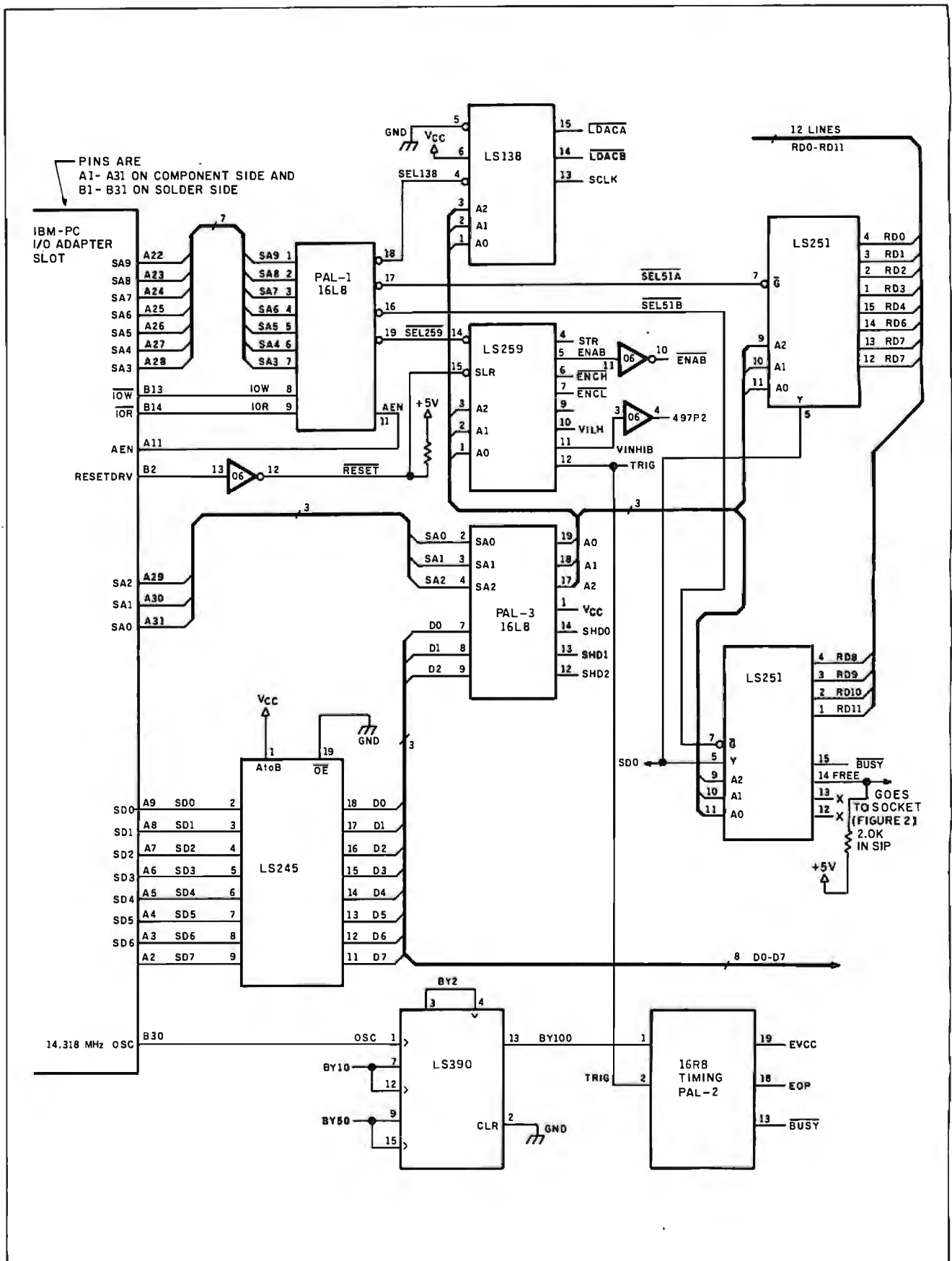


Figure 6: Schematic of the IBM PC interface to ZAP-A-PAL.

**Table 2: I/O address map.** Sixteen consecutive locations are required out of the IBM PC's address space. I chose hexadecimal 100 through 10F, but you can easily change this by modifying the address decoder PAL (PAL-1).

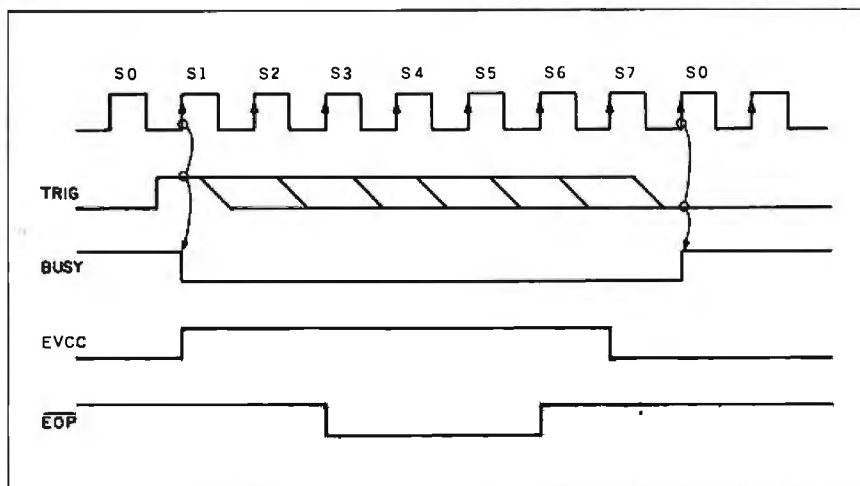
Hex I/O Address	Input	Output
100	Pin 22	Load DAC-A
101	Pin 21	Load DAC-B
102	Pin 20	SCLK—write data to shift register
103	Pin 19	---
104	Pin 18	---
105	Pin 17	---
106	Pin 16	---
107	Pin 15	---
108	Pin 23	Strobe shift data into latches (pulse STR)
109	Pin 14	---
10A	TEST vs. 10.0-V ref	ENCH—enable 0D
10B	TEST vs. 2.5-V ref	ENCL—enable CLOCK
10C	BUSY	---
10D	---	VLH—boost: 0=low, 1=high
10E	---	VINHIB—0 = inhibit booster, 1 = enable
10F	---	TRIG—1 = do program cycle

$V_{CC}$  pin (24 or 20) must not be at  $V_{IH}$  for more than 60 microseconds at a time and with less than a 20 percent duty cycle. Also, the programming pulse on the output pins must be less than 50  $\mu$ s and nominally 20  $\mu$ s long. It is possible to meet these timing constraints using software timing loops, but I don't recommend it. Besides, these requirements provide an excellent opportunity to use a PAL for a finite state machine to control the critical timing of the ZAP-A-PAL board.

A 74LS390 chip divides the 14.31818-megahertz oscillator frequency from the IBM PC bus by 100, yielding a square wave with a period of 6.9841  $\mu$ s. This becomes the clock input of a PAL 16R8 (PAL-2) that acts as a finite state machine to generate two signals: EVCC and EOP. See the timing diagram in figure 7. EVCC drives an IRFD 9123 P-channel HEXFET power MOSFET transistor via a 7406 open-collector inverter. The HEXFET pulls up the  $V_{CC}$  pin of the target PAL from its normal 5-V level to  $V_{IH}$  to apply programming power to the PAL. EOP enables the source driver outputs of a UCN5810A for the duration of the programming pulse to apply  $V_{IH}$  to the selected output pin. The PAL 16R8 also has a BUSY output that can be read via a 74LS251 at I/O address 10C to signal that an operation is in progress. A program pulse is initiated by asserting the TRIG input of the PAL 16R8 by writing a 1 in bit 0 at address 10F. Within 10  $\mu$ s, BUSY will be asserted. The program asserts TRIG, then waits for BUSY to be asserted. The program then clears TRIG and waits until BUSY clears, indicating that the pulse is complete.

There is a classic chicken-and-egg paradox here: You need to program PALs to construct the PAL programmer, but you can't program PALs until the project is done. Fortunately, each circuit where

*continued*



**Figure 7: Timing waveforms of PAL-2, finite state machine.**

**Listing 1: CUPL code to program PAL-3 to protect against conflicts at the pin drivers.**

```

/*****
/* PAL-3 - This device protects against conflicts at the Pin Drivers. */
/* Also buffers A0, A1, and A2. It can be replaced by an LS245 */
/* Allowable Target Device Types: PAL16L8 */
/*****

pin [1..20] = [P1,SA0..2,P5,P6,SD0..2,GND,P11,SHD2..0,!P15..16,A2..0,VCC] ;

A0 = SA0 ;           These just buffer the address lines
A1 = SA1 ;
A2 = SA2 ;

SHD0 = SD0 & !SD1 & !SD2 ;   These protect against conflicts
SHD1 = SD1 & !SD0 & !SD2 ;

```

### Address Decoder PAL expressed in CUPL Logic Design Language

```

/*****
/* PAL-1 - This device decodes I/O addresses to provide strobes for the */
/* following chips: LS259, LS138, LS151(A), LS151(B) */
/* Allowable Target Device Types: PAL16L8 */
/*****

pin [1..20] = [A9..A3,!IOW,!IOR,GND, /* Pin List */
              AEN,P12..15,!SEL51B,!SEL51A,!SEL138,!SEL259,VCC] ;

field IOADR = [A9..A3] ; /* Address Field Spec. */

SEL138 = IOW & !AEN & IOADR:[100..107] ; /* Logic Equations */
SEL259 = IOW & !AEN & IOADR:[108..10F] ;
SEL51A = IOR & !AEN & IOADR:[100..107] ;
SEL51B = IOR & !AEN & IOADR:[108..10F] ;

```

Address Decoder PAL, same logic expressed in PALASM Logic Design Language

IF (VCC) SEL259 = /AEN \* A3 \* /A4 \* /A5 \* /A6 \* /A7 \* A8 \* /A9 \* IOW  
IF (VCC) SEL51A = /AEN \* /A3 \* /A4 \* /A5 \* /A6 \* /A7 \* A8 \* /A9 \* IOR  
IF (VCC) SEL51B = /AEN \* A3 \* /A4 \* /A5 \* /A6 \* /A7 \* A8 \* /A9 \* IOR  
IF (VCC) SEL138 = /AEN \* /A3 \* /A4 \* /A5 \* /A6 \* /A7 \* A8 \* /A9 \* IOW

### Address Decoder PAL, same logic expressed as a CUPL Fuse Plot

```

Pin #19
0000 -----
0032 x-x-x-x-x-x-x-x-x-x
Pin #18
0256 -----
0288 x-x-x-x-x-x-x-x-x-x
Pin #17
0512 -----
0544 x-x-x-x-x-x-x-x-x-x
Pin #16
0768 -----
0800 x-x-x-x-x-x-x-x-x-x

```

LEGEND    X : fuse not blown  
          - : fuse blown

**Listing 3:** *CUPL code to program PAL-2, the timing PAL.*

```

/*****
/* PAL-2 - This PAL controls timing for the VCC and output pin pulses. */
/*
/* Allowable Target Device Types:      PAL16R8
/*****/

pin [1..20]      = [clk,TRIG,P3..9,GND,!OE,!P12,!BUSY,!Q0..3,!EOP,!EVCC,VCC] ;

field state = [Q2..0];

$define [S0..7] 'b'[000..111]

```

- continued

```

sequence state {
    present S0      if TRIG next S1 out BUSY out EVCC ;
                    default next S0 ;
    present S1      next S2 out BUSY out EVCC ;
    present S2      next S3 out BUSY out EVCC out EOP ;
    present S3      next S4 out BUSY out EVCC out EOP ;
    present S4      next S5 out BUSY out EVCC out EOP ;
    present S5      next S6 out BUSY out EVCC ;
    present S6      next S7 out BUSY ;
    present S7      if !TRIG next S0 ;
                    default next S7 out BUSY ;      }

```

PALs are used in this design can be constructed using other components until the ZAP-A-PAL is usable.

With a little extra work, you can bootstrap yourself up to a usable and fail-safe design. You can replace the address decoder PAL (PAL-1) with gates and inverters. You can replace the timing PAL (PAL-2) with a circuit using one-shots, or counters, to produce similar timing pulses. The PAL 16L8 used to filter the shift data and buffer the address lines (PAL-3) is wired so that it can be replaced by a 74LS245 plugged into the same socket. The CUPL files for these three PALs are included in listings 1, 2, and 3 for those who have access to a PAL programmer. Listing 2 shows the equation expressed in both CUPL and PALASM terminology.

### Programmable Voltage Generators

The two programmable voltage generators each consist of an octal 74LS273 register driving a DAC-08 8-bit digital-to-analog converter. Each DAC then feeds into an LM-324 operational amplifier (op-amp) section that acts as a current-to-voltage converter. Each op-amp output drives an adjustable voltage regulator that supplies the high current needed during programming, but at a voltage precisely controlled by the DAC. The feedback resistors on the op-amps determine the full-scale voltage that the circuit produces. Since the DAC current is software-programmable, the voltage out of the op-amp is also software-programmable. To compute the desired value of the feedback resistor, divide the desired full-scale output voltage by the maximum current from the DAC.

DAC-A generates the  $V_{CC}$  voltage for the PAL. The maximum voltage needed for this pin is around 10 V. The feedback resistor for DAC-A can be somewhere around 5.6 kohms to give a full-scale output of 10.9 V.

DAC-B generates the  $V_{IHH}$ , which is the programming voltage for the PALs. The full-scale range for DAC-B is set at around 20 V by a feedback resistor made of two 5.11-kohm resistors in series. The plan here is to use a resistor DIP containing closely matched individual resistors in place of all the 5.11-kohm discrete resistors used in the wire-wrap prototype. As with the reference voltages on the logic verify circuitry, the ratio of the values of the resistors is important, not the absolute values chosen.

### Calibration and Setup

During initial checkout, you should make some measurements to determine the voltage offset between the output of the DAC-B voltage generator,  $V_{IHH}$ , and the actual pins of the PAL socket. The drop across the forward biased diode that connects pin P12 to the ground pin of the PAL you're using ( $gnd\_drop$ ) should also be measured under load. The diode compensates for the saturation voltage of the sink drivers so that a logic low is really near 0 V when referenced to the PAL's GND pin. This drop should be between 0.5 and 0.8 V. These offsets should be included with the value applied to the DAC to compensate for drops in the drivers and the diode.

Calibrating the programmer requires finding the values needed for each DAC to cause a transition on the comparator output when the output crosses the 2.5-V and 10-V reference voltages. Knowing the DAC value at two voltage points lets you calculate the slope of the line in a plot of voltage versus DAC setting. You can then use this plot to find the setting required to generate any voltage within the range of the DACs. You can do all this by software or manually.

Alternatively, you can ignore the reference voltages and use a voltmeter to

measure from P12 to P24 of the PAL socket while stepping the digital input value to DAC-A until you get the desired voltage; then take note of this setting for later use. DAC-B is similarly calibrated by measuring  $V_{IHH}$  between P12 and one of the output pins with the shift register being loaded to activate that pin and a junk PAL (or resistor) in the socket to provide a realistic load on the driver.

The actual value of the reference voltages should be measured at least once with a voltmeter and the values entered into the program as `actual_10V` (for 10 V) and `actual_2P5` (2.5 V). If you don't have a voltmeter, just enter the nominal values of 2.5 V and 10 V, and you won't be too far off.

The AutoCal subroutine in ZAPAL.C first finds the DAC setting for each DAC corresponding to the transition point at each of the two reference voltages. These transition point values are called `DAC_A_low`, `DAC_A_high`, and `DAC_B_low`, `DAC_B_high`. The program then calculates the slope of a line between the two points and, using the DAC setting at one of the reference voltages, has all it needs to know to compute the DAC setting for any other voltage, assuming that the DAC output is linear. The slope of the line connecting the two reference points is calculated as follows:

```

slope_a =
    (actual_10V - actual_2P5)
    / (DAC_A_high - DAC_A_low);
slope_b =
    (actual_10V - actual_2P5)
    / (DAC_B_high - DAC_B_low);

```

Then, when you want to find the DAC setting to yield any wanted voltage within the range of the DAC, you can compute it using the following C code:

*continued*

**Listing 4:** A code extract from ZAP-A-PAL's driver program, ZAPAL.C, showing how you set up the data in the shift register to address a particular fuse.

```

/*      ZAPAL.C - Byte Magazine ZAP-A-PAL Programmer for IBM PC      */

/*      Version 1.9 - (C) by Robert A. Freedman - 23 Oct 1986 - 8:00 PM */

#define base 0x100
#define DAC_A base+0
#define DAC_B base+1
#define SCLK base+2

#define STROBE base+0x8
#define ENAB base+0x9      /* Enable BIMOS drivers */
#define ENCH base+0xA
#define ENCL base+0xB
#define VLH base+0xD
#define VINHIB base+0xE
#define TRIG base+0xF

#define BUSY      (~inportb(base+0xC) & 1)

static int verpin,vad,fuse;      /* Pin # to verify, I/O adr, State */
static int veradr[10] = {9,7,6,5,4,3,2,1,0,8}; /* Mux adr for Pins 14 - 23 */

uchar pins[32]; /* Set up pin values here, then shift out to hardware */

/*      Outputs      Control      Inputs
      2 2 2 1 1 1 1 1 2 1      1      1 1
      2 1 0 9 8 7 6 5 3 4      1 3      2 3 4 5 6 7 8 9 0 1      */
static uchar clear[28] =
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; /* Clear */
static uchar odlo[28] =
    {0,0,0,0,0,0,0,0,0,0,0,0,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; /* OD lo */
static uchar odhi[28] =
    {0,0,0,0,0,0,0,0,0,0,0,0,1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; /* OD hi */

static int pind[24] = /* Maps Pin numbers to Shift Register Position */
    {10,18,19,20,21,22,23,24,25,26,27,28,11,9,7,6,5,4,3,2,1,0,8};

int      n,l,lr,ix,ino,af,T20,fuzno;

int do_a_fuz(fuzno) int fuzno; /* Set up to read or write a fuze */
{
    int half,pin;
    pin = ( fuzno / ( T20 ? 32 : 40 ) ); /* Product-Line # */

    outportb(ENAB,0); /* Disable BIMOS drivers */
    outportb(ENCL,1); /* Disable BIMOS drivers CLOCK */
    outportb(ENCH,1); /* Disable BIMOS drivers OD */

    half = ( T20 ? 32 : 40 ); /* Set OD and CLOCK pins */
    pin( 1, (pin >= half?1:2) ); pin(13, (pin >= half?2:1) );

    (pin >= half ? ldsr(odhi) : ldsr(odlo) ); /* Shift OD & Clock */

    outportb(ENCL,0); /* Enable BIMOS drivers CLOCK */
    outportb(ENCH,0); /* Enable BIMOS drivers OD */

    selfuz(fuzno); ldsr(pins); /* Set up and load Shift-registers */

    outportb(ENAB,1); /* Enable BIMOS drivers */

    return( verifuz() ); /* Read and return state of addressed fuze */
}

zot() /* TRIGger the timing PAL to zap the fuze */
{
    while ( BUSY ) { outportb(TRIG,0); };
    while ( !BUSY ) { outportb(TRIG,1); };
    while ( BUSY ) { outportb(TRIG,0); };
    return(0);
}

```

```

}
int verificuz() /*      Return state of fuze      */
{
    /* Assume the shift-registers are all set up by selfuz(fuzno); */
    outportb(ENCL,1); /* Pulse CLOCK pin by floating */
    outportb(ENCL,0); /* CLOCK to Z momentarily */

    vad = veradr[verpin-14] + base; /* Compute Mux adr of Pin */
    fuse = inportb(vad) & 1; /* Read the state of the fuse */
    /* On 16L8, 16R8 etc PALs, 0 = Blown, 1 = Intact fuse */
    return(fuse);
}
selfuz(fuzno) int fuzno; /* Analyzes fuze-number and sets up all pins */
{
    int an, half, of, ox, lino, pl, pin, i;

    half = ( T20 ? 32 : 40 ); /* T20 is true for 20, false for 24 pin PAL */

    ldsr(clear); /* Clear out old fuze info */

    /*      Compute and place input pins      */

    lino = ( fuzno % half );

    pin = ( fuzno / half );
    if (pin > (T20?63:79) ) return(ERROR);

    lr = 0; if (lino & 2) lr = 2; /* Find which half */
    ix = 0; if (!(lino & 1)) ix = 1; /* Find the state of Pin x */

    /* Now find where to put the selected input pin, ie [I0..I9] */
    for (i=0; i<10; i++) pin(2+i, 2); /* Pull all input pins to VIH */
    ino = lino / 4; pin(2+ino, ix); /* Then set Selected pin to TTL */

    /*      Compute and Place Output Pins      */

    pin( 1, (pin >= half?1:2) ); /*      Set OD and CLOCK pins */
    pin(13, (pin >= half?2:1) );

    pl = pin; if (pin >= half) pl = pin-half;
    an = pl % 8; /* A0..An = pl mod 8 */
    ox = (pl / 8) & (T20?0xF:0x1F); /* Select Outp Pin to pulse */
    for (i=14; i<=23; i++) { pin(i, 0); }; /* Clear all Outputs */

    af = (T20?16:15 ); of = (T20?22:23 );
    if (pin >= half) { af = (T20?19:19 ); of = (T20?18:18 ); };
    if ((pin < half) && !T20) an = bitinv(an, 4);
    an = an & (T20? 7 : 0xF );

    for ( i = (T20?2:3); i >= 0; i--) { /* Set Address bits */
        pin(af+i, ( an % 2 ? 2 : 0 ) ); an = an / 2; };

    pin(of-ox, 4); /* Set Output Pin to Pulse */
    verpin = of-ox; /* Save pin to verify fuse state */
    pin( (pin < half ? (T20?15:14) : (T20?22:23) ), lr); /* Set L/R */
    /* Now all the pins are set for programming or verification */

    int pin(n, val) int n, val; /* Read or Store value of a pin */
    {int v; uchar *p; if (n == 0 || n > 24 ) val = 0xE;
        p = pins + *(pind + n - 1 );
        v = *p; *p = val; return(v);
    }

    ldsr(p) char *p; /* Load pins into Hardware Shift Register */
    {
        int i; i=27; while ( i >= 0 ) { outportb(SCLK, p[i--]); };
        outportb(STROBE, 1); /* Strobe all bits into BiMOS latches */
        outportb(STROBE, 0);
    }
}

```

**Listing 5: An annotated JEDEC file.**

```

CUPL          2.11a Serial# 2-99999-001
Device        p1618 Library DLIB-e-22-8
Created       Thu Oct 09 11:42:26 1986
Name          ZAPAL1 PAL Programmer Address Decoder.
Partno       #12345
Revision      01
Date          8/ 1/86
Designer      Angus Boole
Company       Bovonics Ltd.
Assembly     BYTE ZAPAL programmer
Location      4F
*QP20         ; Specifies the number of pins on the PAL
*QF2048       ; Specifies the number of fuses in the array
*G1           ; Specifies the state of the security fuse
*F0           ; Default state for fuse links not defined below
*L0000 11111111111111111111111111111111 ; Each digit represents a fuse
*L0032 0110101110111011101110111011110 ; 0 means fuse NOT blown
*L0256 11111111111111111111111111111111 ; 1 means fuse IS blown
*L0288 0110101110111011101110111011110
*L0512 11111111111111111111111111111111
*L0544 0110101110111011101110111011110
*L0768 11111111111111111111111111111111
*L0800 0110101110111011101110111111010
*C1C08                                     ; 16 bit checksum of fuse bits

```

$vcc\_want = nominal + gnd\_drop;$   
 $vihh\_want = nominal + gnd\_drop$   
 $+ offset;$   
 for example:

$vcc\_want = 5.00 + gnd\_drop;$   
 $vihh\_want = 11.75 + gnd\_drop$   
 $+ offset;$

$dac\_a = vcc = DAC\_A\_low$   
 $+ (vcc\_want - actual\_2P5)$   
 $/ slope\_a;$

$dac\_b = vihh = DAC\_B\_low$   
 $+ (vihh\_want - actual\_2P5)$   
 $/ slope\_b;$

It should be noted that the range of the DACs does not go to 0 V, and that they are not the same. If you set a DAC to a voltage outside its range, it will clamp, and the output will be inaccurate. The range of each DAC is determined by the value of the feedback resistors around the LM-324 op-amps that were chosen to encompass the needed voltages without dissipating excess power in the LM-317s. The voltage ranges for each DAC are

DAC-A range:  $2.0 \leq V_{cc} \leq 11.0$   
 DAC-B range:  $2.0 \leq V_{ihh} \leq 15.0$

**Programming Algorithm**

The procedure for blowing a fuse is as follows. Load DAC-A with the correct

voltage for the  $V_{cc}$  pin. Load DAC-B to set the voltage to  $V_{ihh}$ . Load the shift registers with all pins set to 0 except the OD pin to 2 and the CLOCK pin to 1. Pulse the strobe (STR) line to load the data latches. This will set OD to  $V_{ihh}$ . Load the shift registers with the proper values for the input ( $I_0$  through  $I_7$ ) and address ( $A_0$ ,  $A_1$ ,  $A_2$ ) lines with the L/R pin specified, keeping OD as before. Also, set the selected output pin driver to a value of 4. Again, pulse the STR line to load the data latches.

To program the fuse, write 1 to the TRIG latch, wait until BUSY is asserted (or at least 10  $\mu s$ ), then write 0 to TRIG. Wait until BUSY is cleared. You can verify the bit by pulsing the CLOCK pin from low to high and then to low again. This is done by pulsing ENCL. The logic level at the selected output pin may be read as bit 0 at the proper I/O address for that pin (check the input section of table 2). If the level is not correct, you can zap the fuse four more times, reading it after each try and waiting between tries so as not to exceed the 20 percent duty-cycle restriction.

When you are done with that fuse, load the shift registers with all pins to 0 except the OD pin to 2 and the CLOCK pin to 1. Pulse the STR line to unload the data latches. This will keep OD at  $V_{ihh}$ . Either select the next fuse by loading a new input line and address as above, or load the

shift register with all 0s to exit from program/verify mode. A verify operation is as above, except that you don't assert TRIG to blow the fuse. See listing 4 for a code example.

**Software Description**

I wrote the ZAPAL program to support an interface between commercially available logic design software and the ZAP-A-PAL hardware. The JEDEC format file is produced as the output of a logic design compiler, such as CUPL or PALASM version 2. The JEDEC file contains a fuse map of the target PAL, parametric information, and some documentation (see listing 5).

**Support Available**

See my article, "Getting Started with PALs" on page 223 for details on obtaining logic design software.

I am preparing a printed circuit board that I am willing to make available at nominal cost to those who wish to build ZAP-A-PAL. I do not intend to offer a kit of parts for this project, but if you cannot find a particular component, I will try to help you out. I will make an assembled board available for those who lack the time or resources to build one for themselves. I am interested in communicating with anyone who has questions about this project or who has found a way to improve on my design. ■