

CIRCUIT CELLAR **I N K**®

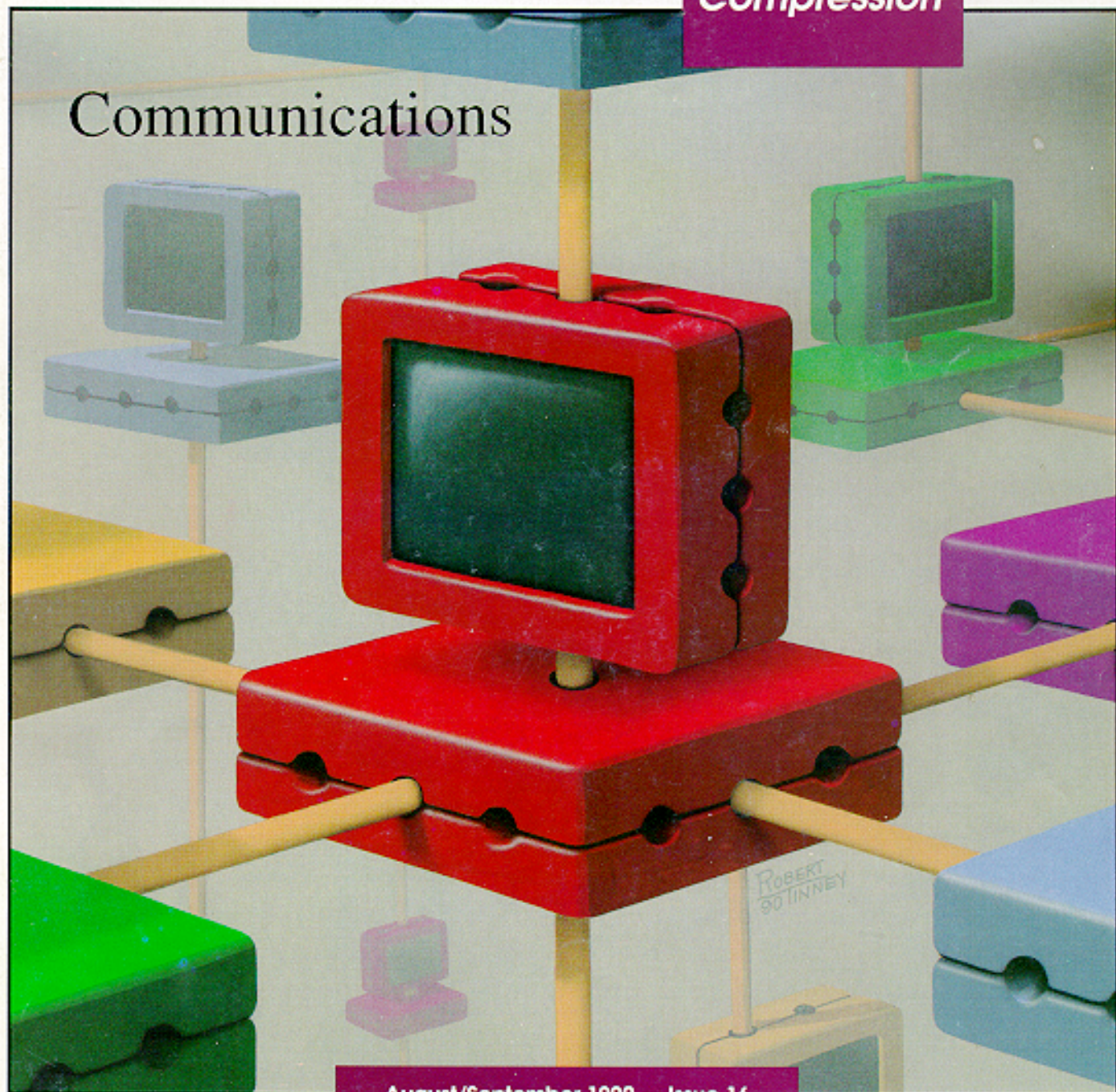
THE COMPUTER  
APPLICATIONS  
JOURNAL

Remote  
PC Control

8096  
Serial  
Expansion

Image  
Compression

Communications



August/September 1990 — Issue 16

\$3.95



0 74470 75349 0

## EDITOR'S INK

Curtis Franklin, Jr.

# A Little Communicating

Ten years ago, networking (in the electronic, not the yuppie, sense of the word) was a wildly futuristic topic. To be certain, large corporations were linking far-flung operations over lines leased from a monolithic Bell System, and some microcomputer users were using 300-bps modems to commit remote computing, but a touch-tone telephone represented state-of-the-art communications to most of the population.

Today, digital networks are a vital part of our society and its work. I don't think of myself as hanging out over the ragged leading edge of technology, but my typical workday would change radically were it not for various networking and communications applications.

I can't imagine trying to **live my** life the way I currently live it without the communications technology. What's more, I'm convinced that the 1990s will see far more changes in technology that drives the way we work than did the 1980s. Let me give you an example.

In 1984, I heard Phil Lemmons (then editor-in-chief of **BYTE**) give a speech in which he predicted that the power of laser printers, scanners, fax machines, and computers wouldn't be realized until they all worked together in a seamless fashion. Products have come around since then that patched two or more of the pieces together, but Phil's vision is still unrealized. Fortunately, silicon is beginning to catch up to imagination in this area. National Semiconductor, among others, has introduced microprocessors that are optimized for controlling laser printers, fax machines, and scanners. As engineers design new applications around these chips, the cost and functionality trends we've seen in the last ten years should continue and, perhaps, accelerate. As I write this, cellular modems and cellular fax machines are considered high-priced executive "toys." Two years ago, cellular telephones were in the same category, but now you can buy a cellular telephone for less than \$100. We're closer than most people believe to portable data appliances that will combine computer, fax, and modem in a four-pound cellular-communicating notebook. When I think about this possibility, and throw in advances in CD-ROM, scanner, and laser-printer technology, my mouth starts to water.

The most important challenge in all of this change is maintaining the *value* of communication. As recently as 100 years ago, the arrival of a hand-written letter was cause for excitement. Fifty years ago, long distance telephone calls or telegrams were special events in most folks' lives. I can still remember **when** an overnight mail package or fax document showing up on your desk was cause for an

immediate interruption in **whatever you** were doing. Now, of course, all of these (with the exception of the handwritten letter—they're still pretty special) have taken their place in a daily routine. Technology and society have given us enormous total bandwidth for communicating. If we can use the bandwidth wisely and avoid trivializing the opportunities it presents, then the millennium has great promise, indeed.

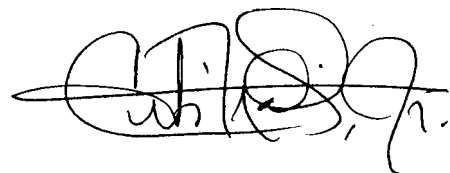
### SPEAKING OF OPPORTUNITIES...

Our readers don't send us a lot of letters. Many of the letters we do receive are in reaction to something we've published. A few letters ask technical questions. Quite a number ask which tools our engineers use. There are great piles of development hardware and software systems available to you, and most of them cost a fair amount of money. It's natural to want to get a recommendation from someone you know before you start writing checks.

Next year, we're going to offer some of those recommendations. We won't print standard reviews, where a professional writer gets to spend a couple of weeks with a product before listing its features. We will let a working engineer or programmer run the product through its paces on nontrivial jobs, and then tell you what it's like to live with the results.

We won't take any pages away from the regular articles in **CIRCUIT CELLAR INK** to print these evaluations. All of them will be contained in special sections, published in addition to the regular issue. I know that this is a subject of special concern to many of you, and I want to personally assure you that we are not going to abandon projects and tutorials in order to talk about products.

We are going to need some help. If you are a working engineer or programmer, and you would like to take part in a product evaluation, please write to me. Tell me about yourself, with an emphasis on your technical qualifications. I need to know what tools you use now, and what your hardware and software setups are. If you've written before, that's great, but we're looking for folks who can thoroughly wring-out an ICE or compiler and tell the **CIRCUIT CELLAR INK** readers where its warts and beauty marks lie. It won't be easy, but it certainly won't be boring.



FOUNDER/  
EDITORIAL DIRECTOR  
*Steve Ciarcia*

PUBLISHER  
*Daniel Rodrigues*

EDITOR-in-CHIEF  
*Curtis Franklin, Jr.*

PUBLISHING  
CONSULTANT  
*John Hayes*

ENGINEERING STAFF  
*Ken Davidson*  
*Jeff Bachiochi*  
*Edward Nisley*

CONTRIBUTING  
EDITOR  
*Thomas Cantrell*  
*Christopher Ciarcia*

NEW PRODUCTS  
EDITOR  
*Harv Weiner*

CONSULTING  
EDITORS  
*Mark Dahmke*  
*Larry Loeb*

CIRCULATION  
COORDINATOR  
*Rose Manse/la*

CIRCULATION  
CONSULTANT  
*Gregory Spitzfaden*

ART & PRODUCTION  
DIRECTOR  
*Tricia Dziedzinski*

PRODUCTION  
ARTIST/ILLUSTRATOR  
*Lisa Ferry*

BUSINESS  
MANAGER  
*Jeannette Walters*

#### STAFF RESEARCHERS

Northeast  
*Eric Albert*  
*William Curlew*  
*Richard Sawyer*  
*Robert Stek*

Midwest  
*Jon Elson*  
*Tim McDonough*

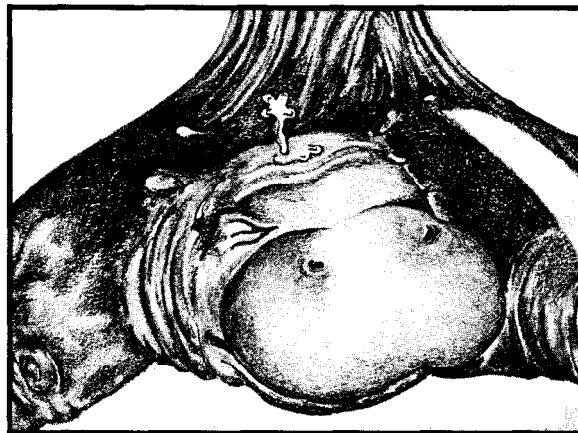
West Coast  
*Frank Kuechmann*  
*Mark Voorhees*

Cover Illustration  
by Robert Tinney

# CIRCUIT CELLAR **INK**<sup>®</sup>

## THE COMPUTER APPLICATIONS JOURNAL

# In This Issue...



### **18** Image Compression for High-Speed Network Transmission by Chris Ciarcia

When you have huge (and growing!) image files and a finite bandwidth over which to move them, something has to give. A good image compression method can stretch your network capabilities while preserving usable image quality.

### **28** Extended Serial Communications on the 8096 *Increase the Utility of these Ubiquitous Chips with Simple C Software* by Alfred L. Schumer

Intel's 8096 is a powerful microprocessor with considerable I/O muscle—but a simple software tune-up can coax even more serial horsepower from this popular chip.

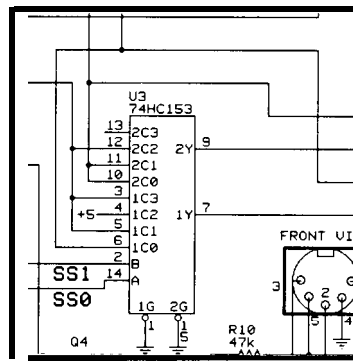
## DEPARTMENTS

Editor's INK		
A Little Communicating by Curtis Franklin, Jr.	_____	1
Reader's <b>INK</b> —Letters to the Editor	_____	5
NEW Product News	_____	12
Firmware Furnace		
The Furnace Firmware Project Keypad and Piezo Beeper by Ed Nisley	_____	56
From the Bench		
Creating a Nonvolatile RAM Module by Jeff Bachiochi	_____	65

**34** ONDI-The ON-line Device Interface  
**Building a Powerful Remote Control for your PC**

by John Dybowski

You've seen "remote control" software that lets you manipulate another computer from your keyboard. This low parts-count device provides full remote control (including AC power control) and security for complete remote computing.



Circuit Cellar BBS-24  
 Hrs. 300/1200/2400 bps, 8 bits, no parity, 1 stop bit,  
 (203) 871-1988.

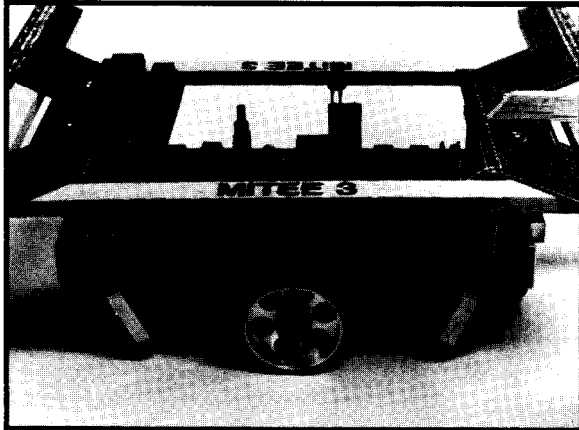
The schematics provided in Circuit Cellar INK are drawn using Schema from Omation Inc. All programs and schematics in Circuit Cellar INK have been carefully reviewed to ensure that their performance is in accordance with the specifications described, and programs are posted on the Circuit Cellar BBS for electronic transfer by subscribers.

Circuit Cellar INK makes no warranties and assumes no responsibility or liability of any kind for errors in these programs or schematics or for the consequences of any such errors. Furthermore, because of the possible variation in the quality and condition of materials and workmanship of reader-assembled projects, Circuit Cellar INK disclaims any responsibility for the safe and proper function of reader-assembled projects based upon or from plans, descriptions, or information published in Circuit Cellar INK.

CIRCUIT CELLAR INK (ISSN 0896-8985) is published bimonthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (203) 875-2751. Second-class postage paid at Vernon, CT and additional offices. One-year (6 Issues) subscription rate U.S.A. and possessions \$14.95, Canada/Mexico \$17.95, all other countries \$26.95-surface, \$38.95-air. All subscription orders payable in U.S. funds only, via international postal money order or check drawn on U.S. bank. Direct subscription orders to Circuit Cellar INK, Subscriptions, P.O. Box 3050-C, Southeastern, PA 19398 or call (215) 630-1914.

POSTMASTER: Please send address changes to Circuit Cellar INK, Circulation Dept., P.O. Box 3050-C, Southeastern, PA 19398.

Entire contents copyright 1990 by Circuit Cellar Incorporated. All rights reserved. Reproduction of this publication in whole or in part without written consent from Circuit Cellar Inc. is prohibited.



**40** Building MITEE Mouse III  
 Part 2-The Software for a Maze-Running Rodent

by David Otten

In the last issue, we built a robotic mouse — now we show it how to find its way in and out of a maze. Motor position control, mapping, and diagonal negotiation are all here-and with no cheese in sight.

**53** Huge Arrays on the HD64180  
 Taking Advantage of Memory Management

by Jack Ganssle

Hitachi's HD64180 becomes even more powerful with software techniques for using very large data arrays.

Advertiser's Index	73
Silicon Update	
Old 8051s Never Die-They Just Get Smarter	74
<i>New Power for a Controller Mainstay</i>	
<i>by Tom Can trell</i>	
Practical Algorithms	
Getting to Know You	79
<i>A New Feature Begins</i>	
<i>by Scott Robert Ladd</i>	
<b>ConnecTime</b> — <i>Excerpts from the Circuit Cellar BBS</i>	82
<i>Conducted by Ken Davidson</i>	
Steve's Own INK	
Flash or Splash?	88
<i>by Steve Ciarcia</i>	

# letters to the Editor

## HOLOGRAMS

After reading the article on computer-generated holograms (**CIRCUIT CELLAR INK #14**), I decided to try it on my AT. It's the old 6-MHz model, without a coprocessor, running QuickBASIC. According to some timing checks I did on the program, it would take about 1836 seconds per column to display an interference pattern. That translates to about 363 hours, or 13 days for an entire hologram. Although I have a certain amount of patience, I don't think I have that much. I decided, instead, to see if I could speed up the process some. The included program (Listing 1) is my latest effort. The program runs about 8.7 times faster than the original, taking about 210 seconds per line or 37 hours per image. Actually, since I am running an EGA monitor and can use only 200 vertical points anyway, I can cut it down to 105 seconds per line, or 18 hours per screen.

The basic idea behind the speed-up was to get the trig functions out of the inner loop and to use the `SQR` function instead of the `LOG` function used by the `"^"` operator. Trig and log functions are painfully slow without a coprocessor, and that is where virtually all of the speed-up occurs. The use of integers in the loop variables also helps, but the speed-up is more subtle.

Although the trig functions probably don't have to be double precision, `SQR` does. For those BASICs which don't have double-precision `SQR` functions, you can use Newton's method, instead. If you use the original Z distance as the first guess, you can get by with just two iterations of the function. For those who don't remember,

```
b=d/2:      ' first guess, using original Z
b=(d/d+b)/2: ' first iteration
b=(d/b+b)/2: ' second iteration
d=b:       ' "return" square root
```

Also, I use the `FIX` function which returns the fractional part of a floating-point number. For those who don't have that function, you can use `INT` if you reduce the size of `d` first. The following should work:

```
d=d-zd:    ' subtract off original Z
w=d/l:     ' find wavelength multiple
i=w/int(w): ' get rid of integer part
```

Finally, the hologram appears to be the intersection of an X-Y plane (the hologram itself) with the concentric

hemispheres formed by the interference of the two waves. Also, any given source point is inherently distinguishable from any other. If this is true, then all you need to do is calculate all of the interference hemispheres for one point, and perform a "look it up in the table" process for all the other points. To reduce the size of the table, you can restrict the Z range and map all `-Xs` and `-Ys` to their positive counterparts. A 3-D camera can be made with two CCD devices which are set apart with a program to figure out the X,Y,Z coordinates of the corresponding points of the images. These coordinates can be passed to the hologram maker which can produce the hologram. Not exactly a weekend project, but fun to think about. .

I'm glad you printed the article. I learned a lot from it. It took me a while to figure out how the program worked; it's been a LONG time since I've been in a physics class. Still, it was fun to make it faster.

Richard F. Brown  
Oakhurst, CA

My son and I really enjoyed the "Computer-Generated Holographic Images" article in your April/May 1990 issue. We've made a few holograms, and are discovering new things to try with each one. Some observations:

What the article suggests seems dangerous to us. The article doesn't say precisely how to view the holograms, but does say, "Also, the distant virtual image of the rose can be seen by looking through the hologram toward the illuminating laser." Since the image "surrounds" the illuminating laser beam, this means looking (nearly) directly into the laser beam. We're not experts, but that doesn't seem safe to us.

We decided to use an HP LaserJet IIP printer to make the hologram. This allowed us to quadruple the resolution (1920 x 2560). We made the overall image the same size on the film. This allowed us to move the "object" closer to the film by a factor of four, making it appear four times as big.

In addition, if one is not quite so conservative, one can move the image closer by another factor of four. True, some of the parts of the image "fade" a little when one looks at some parts of the hologram; Nonetheless, the effect is, we think, much more pleasing.

The resulting holograms can be viewed without a laser! Here's how: get any pinpoint light source (fairly far away). One simple possibility is to just set up a slide projector with no slide in it so that it projects a white screen. (It may help to put aluminum foil with a pencil-sized hole in it over the projector lens to make the light source smaller.) Now look at the light source (e.g., with the projector, stand where the screen should be and look back at the projector). Look through the hologram at the point source. You will often see two images: one smaller and sharp, the other bigger and out of focus. We think that two images are produced: a virtual image "behind" the hologram, and a real image on your side of the hologram.

The real image can be photographed easily. If you have a 35mm camera with a bellows, take the lens off the camera and put the bellows on. Don't put any lens on the camera. Instead, use masking tape to tape the hologram where the lens would go. Now point the camera at the pinpoint source as above. When the hologram is about 6.5" from the film plane you will see a sharp reconstruction of the hologram in the viewfinder.

We are enclosing the C code used to make an "RIT" hologram (see Listing 2). The younger of us is a student at the Rochester Institute of Technology, hence the initials. You'll notice that we have modified the author's code so that the pattern is not recomputed in the innermost loop (as he does with the rose), and we have replaced the sine computation with a simple table look-up. Although we're

doing about 10 times as many computations as the author does, it took only 25 hours to compute the RIT hologram. We think that even bigger speed-ups are possible.

Our printer has an extra one megabyte of memory. If you use a LaserJet II without extra memory the image will be split between two sheets of paper. The bigger image should produce a usable hologram. To get the hologram printed (after you run the program), use the command:

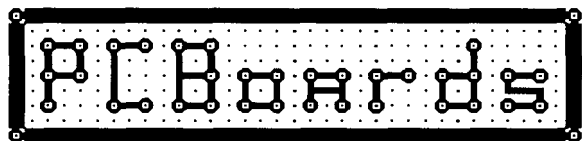
```
copy /b holo1pt1:
```

Thanks for publishing such a stimulating article!

David Heath  
Michael Heath  
Ithaca, NY

*The Messrs. Heath raise a most important point: you should never, under any circumstances, look directly into a laser beam unless you know precisely what you're doing. Serious eye damage can result, even from a low-power laser. In Dale's case, he was viewing the holograms at a far enough distance from the laser (on the order of 50' or more) and with enough divergence of the beam that eye damage was avoided.*

We're excited to hear that **CIRCUIT CELLAR INK** readers are experimenting with the information given in the article. We hope that, as others experiment, you'll let us know about your results.  
Editor



### PC-B ARTWORK MADE EASY !

Create Printed Circuit Artwork on your  
IBM or Compatible

- \* MENU DRIVEN
- \* HELP SCREENS
- \* ADVANCED FEATURES
- \* EXTREMELY USER FRIENDLY
- \* AUTO GROUND PLANES
- \* 1X and 2X PRINTER ARTWORK
- \* 1X HP LaserJet ARTWORK

\* HP and HI PLOTTER DRIVER optional 49.00

REQUIREMENTS: IBM PC or Compatible, 384K RAM  
DOS 3.0 or later. IBM compatible printers.

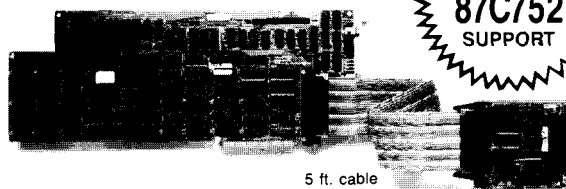
PCBoards - layout program 99.00  
PCRoute - auto-router 99.00  
SuperCAD - schematic pgm. 99.00  
DEMO PKG. - 10.00

Call or write for more information  
PCBoards

2110 14th Ave. South, Birmingham, AL 35205  
(205) 933-1122



"The Best 8051 Emulator"



5 ft. cable

# 8051

SEE EEM 89/90  
Pages D 1324-1326

PC based emulators for the 8051 family

8031, 8032, 8051, 8052, 80C152/154/321/451/452/51FA/515/517/535/537/552/  
562/652/851, 80532, 83C451/552/652/751/752/851, 8344, 87C451/552/751/752,  
8751, 0752, DS5000 + CMOS more.

- PC plug-in boards or RS-232 box.
- Up to 24 MHz real-time emulation.
- Full Source-level Debugger w/complete C-variable support.
- 48 bit wide, 16K deep trace, with "source line trace."
- "Bond-out" pods for 6051, 83C552, 83C451, 83C652, 83C751, 80C515/80C517, 83C752.

Prices: 32K Emulator 8031 \$1790; 4K Trace \$1495'

CALL OR WRITE FOR FREE DEMO DISK!

Ask about our demo VIDEO

**NOHAU** CORPORATION  
51 E. Campbell Avenue  
Campbell, CA 95008  
FAX (408) 378-7869  
(408) 866-1820

'US only

You wanted me to write about "how I think about a problem" so that your readers could "get inside the designer's head." Well, I'm starting that process by taking issue with "A Few Words from the Staff" in *CIRCUIT CELLAR* INK #14.

At one time, I knew a metal artisan who could mill a part more accurately with a tape measure than most

machinists could using a dial caliper. Similarly, I doubt very much that Woz would have needed an oscilloscope to design the Apple I. By the same token, all the test equipment in the entire Tektronix catalog won't turn an idiot into a gifted engineer.

My point is that it's not the equipment solving the problem-it's the person using it. I have a 16-channel, 50-MHz logic analyzer and a 500-MHz, 4-channel 'scope. I also have a \$17.00 Radio Shack logic probe and a brain.

```

DEFDBL A-Z
DIM xc(100)
DIM yc(100)
DIM psin(1000)

pi = 3.1415926535897938
l = 0.025514496%

zd = 129528#
zd2 = zd * zd

hc = 320
vc = 240
s = 0

pr = 252

true% = NOT 0
vga% = NOT true%
IF vga% = true% THEN
  vsize% = 479
  ssize% = 1
  yd% = 1
  SCREEN 12
ELSE
  vsize% = 390
  ssize% = 2
  yd% = 2
  SCREEN 2
ENDIF

PRINT "Creating sine table..."
FOR a% = 0 TO 1000
  psin(a%) = SIN(a% * pi/500)
NEXT
PRINT "Sine table complete.%"

PRINT "Building Rose..."
FOR t% = 0 TO 42
  a = t% * pi / 42
  r = pr * COS(3 * a)
  xc(t%) = hc + r * COS(a)
  yc(t%) = vc + r * SIN(a)
NEXT
PRINT "Rose complete."
CLS
FOR x% = 0 TO 639
  FOR y% = 0 TO vsize% STEP ssize%
    s = 0
    FOR t% = 0 TO 42
      xd = xc(t%) - x%
      yd = yc(t%) - y%
      d = SQR(xd*xd+yd*yd+xd2)
      w = d / l
      w = w - FIX(w)
      i% = INT(1000 * w)
      s = s + psin(i%)
    NEXT
    IF s >+ 0 THEN PSET(x%,y% \ yd%), 1
  NEXT
NEXT
NEXT

```

```

' define all double precision
' x coordinate array for object
' y coordinate array for object
' array of sine values so that they
' don't have to be calculated in
' the loop.
' pi taken to lots of decimal places
' lambda (wavelength) in pixels
' 6328 x 10^10 m x 16 x 2520
' 16 - reduction ratio
' 2520 - pixels per meter
' z distance (in pixels) of image
' 51.4 m x 2520 pixels/meter
' zd squared - saves doing it more
' than once.
' horizontal center of image
' vertical center of image
' sum of interference amplitudes
' at any given point
' polar radius of rose in pixels
' 0.1 m x 2520 pixels/meter
' set up value for true
' not running VGA
' if display is VGA
' set vertical size to 480
' step size of 1 in for loop
' y divider in pset() to 1
' set screen to mode 12
' if not VGA (in my case, EGA)
' vertical size is 400
' set step size to 2 in for loop
' y divider in pset() is 2
' and screen mode is 2
' size 400 divider 2 gets rid of
' most of the distortion of EGA

' figure out a table of sine values
' for one wave length - this is done
' so that sine, a very slow function
' does not need to be done more than
' once - array lookups are faster

' create the rose - 42 points
' a - angle at point t
' r - polar radiu at angle a
' x coordinate of rose point
' y coordinate of rose point

' make sure the screen is clear
' scan across the horizontal
' down the vertical
' set sum to zero
' and through the rose
' x distance
' y distance
' distance to point
' convert to multiple of wavelength
' get rid of the whole number part
' to form index into sine array
' add sine value to interference sum
' end of rose loop

' if sum is positive, put in point
' end of vertical scan loop
' end of horizontal scan loop

```

listing 1 -A coprocessor and some tricky programming speeds up the calculation of computer-generated holograms.

```

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define PI 3.141592653589793
#define ESC '\033'
FILE *fptr;

main()
!
long int index;
double s, l, pz, h, k, a, x, y, r, px[42],
phase, d, twopi, py[42], tempt, twopiavl,
pz2, px1, pyl;
float sintable[100];
int nbits = 0, t, indx;
char bite = '\000';
printf("Go to executable code!\n");
s = 0;
l = 0.0255145;
twopi = 2.0 * PI;
twopiavl = twopi / l;
pz = 129528/16.;
pz2 = pz * pz;
h = 320;
k = 240;
a = 252;
for (t=0;t<=99;t++)
sintable[t] = sin(twopi*t/100);
px[0] = 8;      py[0] = 396;
px[1] = 8;      py[1] = 344;
px[2] = 8;      py[2] = 292;
px[3] = 8;      py[3] = 240;
px[4] = 8;      py[4] = 188;
px[5] = 8;      py[5] = 136;
px[6] = 8;      py[6] = 84;
px[7] = 60;     py[7] = 396;
px[8] = 112;    py[8] = 370;
px[9] = 138;    py[9] = 318;
px[10] = 112;   py[10] = 266;
px[11] = 60;    py[11] = 240;
px[12] = 86;    py[12] = 188;
px[13] = 112;   py[13] = 136;
px[14] = 138;   py[14] = 84;
px[15] = 242;   py[15] = 396;
px[16] = 294;   py[16] = 396;
px[17] = 346;   py[17] = 396;
px[18] = 294;   py[18] = 344;
px[19] = 294;   py[19] = 292;
px[20] = 294;   py[20] = 240;
px[21] = 294;   py[21] = 188;
px[22] = 294;   py[22] = 136;
px[23] = 242;   py[23] = 84;
px[24] = 294;   py[24] = 84;

```

```

px[25] = 346;   py[25] = 84;
px[26] = 450;   py[26] = 396;
px[27] = '502'; py[27] = 396;
px[28] = 554;   py[28] = 396;
px[29] = 606;   py[29] = 396;
px[30] = 658;   py[30] = 396;
px[31] = 554;   py[31] = 344;
px[32] = 554;   py[32] = 292;
px[33] = 554;   py[33] = 240;
px[34] = 554;   py[34] = 188;
px[35] = 554;   py[35] = 136;
px[36] = 554;   py[36] = 84;

if((fptr = fopen("holo", "wb"))== NULL) {
printf("Bad file!\n");
exit(0);
}
putc(ESC, fptr);
fputs("t300R", fptr);
putc(ESC, fptr);
fputs("r0A", fptr);
for(x=0;x<=2559;x++) {
printf("x = %lf\n", x);
putc(ESC, fptr);
fputs("b240W", fptr);
for(y=0;y<=1919;y++) {
for(t=0;t<=36;t++) {
px1 = px[t] - x/4.;
py1 = py[t] - y/4.;
d = sqrt(px1*px1 + py1*py1 + pz2);
phase = d/l;
index = (long int) (phase);
indx = 100.0 * (phase - index);
s = s + sintable[indx];
}
if(s>0) s = 0;
else s = 1;
bite = bite + bite + s;
nbits = nbits + 1;
if(nbits >= 8){
nbits = 0;
putc(bite, fptr);
bite = '\000';
}
s = 0;
}
}
putc(ESC, fptr);
fputs("rB", fptr);
putc('\014', fptr);
fclose(fptr);
}

```

listing in C, the code for generating a hologram of the RIT logo illustrates some time-saving shortcuts.

These latter instruments, I find, are often sufficient for the LSTTL world. Here are some of my suggestions for minimal-test-equipment debugging.

**Understand the circuit**—if you're building a project from an article or a databook, take the time to understand, in detail, exactly how it works. This may mean doing some extra background reading, but then use of the library is free.

If it's your own design that doesn't work, check your assumptions. A little a priori knowledge is a dangerous thing. In the past, I've wasted time because I assumed that a chip worked in a certain way, only to discover that ICs, like computers, do exactly what you tell them to do—not necessarily what you want them to do!

**Use development tools which permit interactive debugging**—If you're writing in a high-level language, use a compiler that does so immediately. This allows you to

make subtle changes in the code and immediately observe their effects.

If you're writing ROMable code, beg, borrow, or steal some type of EEPROM or RAM module into which you can serially load the object code for testing. If you have to constantly erase EPROMs, you're much less inclined to make subtle changes for debugging.

I often find it faster, even when using assembler, to write a test fragment and check functionality with the logic probe, rather than wading through a logic analyzer hex dump.

Furthermore, I believe software simulators are of limited value for observing bottlenecks and worst-case interrupt behavior. In general, it's much better to actually try the code on the target hardware.

**Design-in debugging aids**—LEDs, serial ports, beepers, and so on go a long way toward uncovering your mistakes.



Take a lesson from the software gurus-Design and test the hardware from the bottom up, and use simple "primitives" to build complex assemblies.

Use good construction practices-Don't allow unused pins to float. Install adequate bypass caps. Really understand grounding and shielding techniques.

Sleep on it-1 can't count the number of times I have given up on a problem late at night, only to awaken the next morning and immediately fix it. Your subconscious mind is incredibly powerful-use it! (My only problem is how to bill the client for that time.)

J. Conrad Hubert  
St. Paul, MN

Jim Hubert is a frequent contributor to CIRCUIT CELLAR INK.

## FEEDBACK

I would like to take this opportunity to compliment you on the material contained in CIRCUIT CELLAR INK. Not only do I find the articles technically informative, but they are very practical and help me a great deal in my work.

I would especially like to compliment you on an article you ran in CIRCUIT CELLAR INK #12, "A Low-Cost MIDI Sequencer" by Winefred Washington. This type of infor-

mation is very helpful to us in the north woods of Idaho. I have already used some of Mr. Washington's design on the keyboard display section of a portable data logger project. I hope you will continue to feature more of the Design Contest projects in the future.

Keep up the good work. CIRCUIT CELLAR INK is the most important literature I receive.

Charles J. Mancuso  
Sandpoint, ID

Regarding the letter to the INK research staff in CIRCUIT CELLAR INK #12 that getting an FCC certification

\$3000e is little as product is well designed and passes the certification the first time around. A reputable EMC test lab will run the test for about \$1000, and will write a report for about \$600. FCC form 731 can be filled out by the manufacturer, or by the test lab for an additional \$600. The FCC is \$650. If the test report the equipment passes the FCC limit by a reasonable margin, it is likely that the FCC will not require that the product be sent to their laboratory for a retest, unless it is a personal computer, as defined in Part 15.

As an EMC consultant, I have found that the major reason for the high cost and schedule delays associated with obtaining a certification is that the product is often designed by engineers who are not sufficiently experienced in designing equipment to minimize electromagnetic emissions. The result is that the product fails the test, and must then be redesigned or "fixed," retested, "fixed" again, and so on. If sufficient care is given to the design of the grounding scheme, clock distribution, power distribution, on-board filtering, and packaging, the product is much more likely to pass the certification test on the first try.

## 68000 K-System Factory Direct Prices

Now there is a bus that makes it easy to use the entire family of 68000 components. Utilizing native 68000 signals, the K-Bus makes it possible to create low cost 68000 systems in a straightforward manner. The simplicity inherent in the K-System concept allows the system designer the ability to concentrate on meeting the demands of the applications. This same simplicity combined with its low cost makes the K-System ideal for applications ranging from personal use through educational and Laboratory applications up to industrial control and systems development. All of this is accomplished at no sacrifice in performance or reliability.

The convenient size (4 x 5 1/4 inch) of the K-Bus boards permits the optimal division of system functions thus simplifying system configuration. The motherboard incorporates integral card guides and compatible power connectors which minimizes packaging requirements. Both SKDOS and OS-9/68000 are fully supported allowing efficient system utilization in both single and multi-user applications.

Boards currently in production: **AVAILABLE IN KIT FORM**

<b>K-BUS</b>	12 Slots, .8" centers, PC type power connectors	\$129.95
<b>K-CPU-68K</b>	10MHz 68000 CPU, 2 ROM sockets (12 or 16MHz)	\$129.95
<b>K-MEM</b>	256K static RAM or 27256 type EPROMs (OK installed)	\$ 59.95
<b>K-ACI</b>	2 serial ports with full modem controls (68681)	\$ 99.95
<b>K-FDC</b>	Floppy disk controller (up to four 5 1/4 drives)	\$ 99.96
<b>K-SCSI</b>	Full SCSI implementation using 5380 chip	\$ 99.95
<b>K-DMA</b>	2 channel DMA controller using 68440 chip	\$129.95
<b>K-PROTO</b>	General purpose/wirewrap board	\$ 39.95
<b>K-XXX-BE</b>	Bare board with documentation for above	\$ 39.95

Software:		
<b>SKDOS</b>	Single user, editor, assembler, utilities, BASIC	\$150.00
<b>OS-9/68000</b>	Multi-user, editor, assembler, SCRED, utilities, BASIC, C, PASCAL, FORTRAN are available	\$300.00

Inquire about our UniQuad line of 68xxx Single Board Computers.  
Quantity and package discounts available

Terms: Check, Money Order, Visa, MasterCard—Prices include UPS ground shipment in continental U.S.

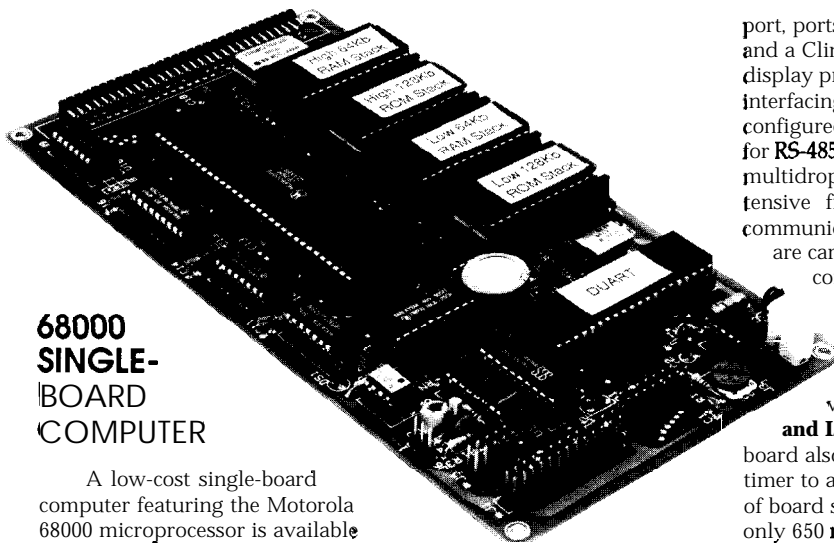
## Hazelwood Computer Systems

Highway 94 at Bluffton UniQuad™ K-Kits™  
Rhineland, MO 65069 • (314) 236-4372

Reader Service #11

Have a quality project  
you've been keeping  
secret?

Tell the world about it by writing  
for Circuit Cellar INK!



## 68000 SINGLE-BOARD COMPUTER

A low-cost single-board computer featuring the Motorola 68000 microprocessor is available from **Vesta Technology Inc.** The computer contains an on-board ROM-resident Forth development environment to facilitate product development.

The 8" by 4" board runs at

either 8 or 16 MHz and provides socketing for up to 128K of battery-backed RAM and up to 256K of ROM. A battery-backed real-time clock, a parallel printer

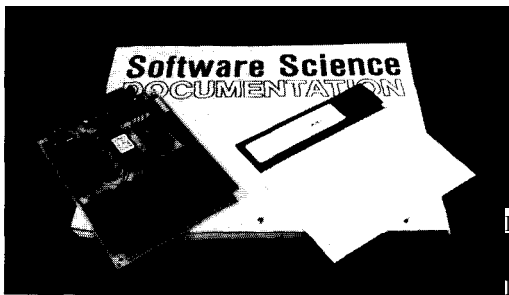
port, ports for a 4 x 5 keypad, and a Cline by 40-character LCD display provide easy application interfacing. Two serial ports, one configured for RS-232 and one for RS-485 with addressable multidrop capabilities, allow extensive firmwaresupported communications. All bus signals are carried to an expansion connector to support expansion boards with a total of 16 megabytes of memory and a variety of A/D, D/A, and I/O configurations. The board also includes a watchdog timer to assure program control of board status, and requires only 650 mA at +5 volts. A negative voltage generator for the RS232 port is provided on-board.

The development language in ROM is a direct-threaded 32-bit (address and data) extended

version of Forth-83 with multitasking and assembler capabilities. The language also provides for autostart of ROM-resident application code. The board requires only an IBM PC and an EPROM programmer for a complete development system. A PC-resident communications package provided with the board (**VestaComm**) permits use of the PC's disk to transparently store and download code and data to the 68000 board while acting as a console to the 68000.

The board sells for \$295 in single quantities and includes 64K of RAM.

**Vesta Technology, Inc.**  
7100 West 44th Avenue,  
Suite 101  
Wheatridge, CO 80033  
(303) 422-8088  
Fax: (303) 422-9800  
Reader Service #213



## Z8 DEVELOPMENT SYSTEM

ProtoQuick Z8 is a microprocessor-based prototyping and application development system. The 4.5" x 6" single-board computer and prototyping board is based on the Zilog Z8 microprocessor chip. Along with nearly 12 square inches of prototype area, ProtoQuick Z8 has EPROM, RAM, RS-232 serial communications, and a decoded S-position DIP switch. Standard 28pin EPROM and RAM sockets support up to 32K of EPROM and 8K of RAM. The Z8 single-chip microprocessor provides six vectored interrupts, two counter/timers, as well as bit, nibble, and byte-wide TTL I/O. The RS-232 interface operates at standard rates up to 38,400 bps and all of the Z8's 14 user-configured I/O lines are available at the prototype area.

Proto-Quick Z8 board needs only a single 5-volt power supply.

Proto-Quick Z8's application development tools include the Software Science Z8

operating system in EPROM, Software Science's ASMZ8 MS-DOS-based Z8 cross-assembler, and Zilog's Z8671 BASIC/Debug BASIC-in-ROM CPU chip.

At \$99.00, ProtoQuick Z8 comes completely assembled and ready to run with the Software Science ASMZ8 MS-DOS cross-assembler and a copy of the complete Zilog Z8 Technical Manual. The ProtoQuick Z8 board with parts list, assembly drawings, operating system in EPROM, and Z8 technical manual is \$39.00. The Z8671 BASIC-in-ROM version of the Z8 chip is available separately for \$19.00.

Software Science  
3750 Roundbottom Rd.  
Cincinnati, OH 45244  
(513) 561-2060

Reader Service #214

## REMOTELY PROGRAMMABLE ROMDISK

A solid-state disk and drive emulator that enables IBM PC/XT/AT and compatible computers to be used as diskless terminals with auto-booting and special safeguards for LANs and other networks is available from Curtis Inc. The ROMDISK FERO includes security features to allow operation of terminals without mechanical disk

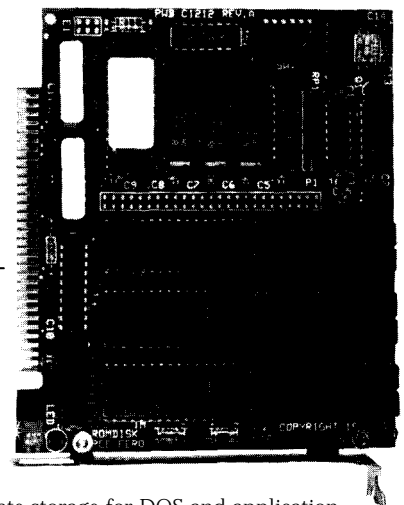
drives and provides solid-state storage for DOS and application programs in the user terminal. LAN utilities enable the unit to be remotely erased and reprogrammed from a supervisory terminal on the network without intrusion of the terminal. The unit can also be incorporated as part of a home security system with a master control system updating remote terminals as required.

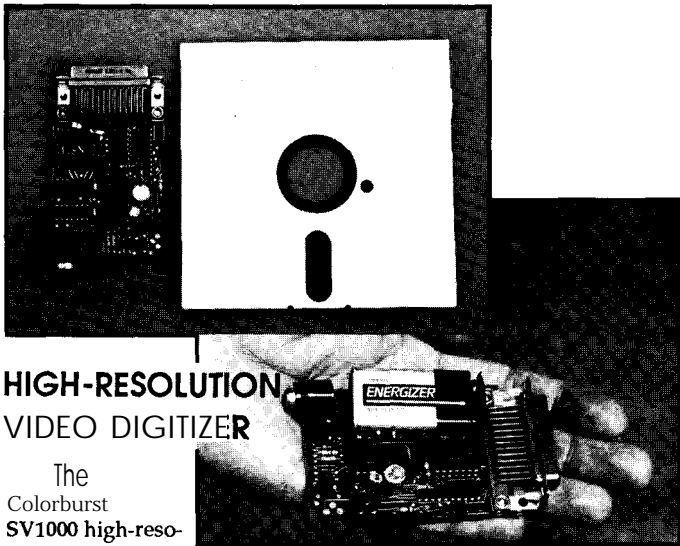
The ROMDISK FERO is capable of emulating standard 3.5" and 5.25" diskettes up to 720K of storage. The unit uses flash EEPROMs that are electrically erasable and programmable or standard ultraviolet erasable EPROMs that have been programmed on a programmable model for read-only operations.

The ROMDISK FERO lists for \$279 with UV EPROMs and \$319 with flash EEPROMs. Other models include the ROMDISK PCE/2 which emulates 3.5" and 5.25" diskettes up to 1.3M, programs EPROMs or flash EEPROMs, and dual operation by emulating a secondary diskette with a battery-backed SRAM daughter board.

Curtis, inc.  
2837 North Fairview Ave.  
St. Paul, MN 55113  
(612) 631-9512 • Fax: (612) 631-9508

Reader Service #215





## HIGH-RESOLUTION VIDEO DIGITIZER

The Colorburst **SV1000** high-resolution video digitizer plugs into the parallel port of any IBM PC, XT, AT, or PS/2 compatible and can capture video pictures from TV cameras, VCRs, or other composite video sources. Applications for the compact 2" by 3" unit include desktop publishing, inspection, computer animation, and pattern recognition.

Two modes, both with 256 gray levels, provide either 640x480 or 320x200 resolution. Pictures can be displayed on CGA, EGA, or VGA monitors, saved to disk with menu-driven software, or exported to other programs via the "grab" or "freeze" utilities included with most desktop publishing and paint programs. Capture time is 10 to 20 seconds, depending on resolution, and the unit runs from any TV camera, VCR, or

video input.

The included software provides simple menu-driven setup, picture load and save routines, and features built-in diagnostics to check both the video digitizer and the input video source. Contrast, brightness, sync, and unit on/off are also controlled by software. The Colorburst **SV1000** sells for under \$90.00 and a packaged version with a printer bypass switch is also available. A demonstration disk is available for \$3.00.

Colorburst  
P.O. Box 3091  
Nashua, NH 03061  
(603) 432-2001

Reader Service # 216

## CUSTOMIZABLE MULTIAXIS INDEXER

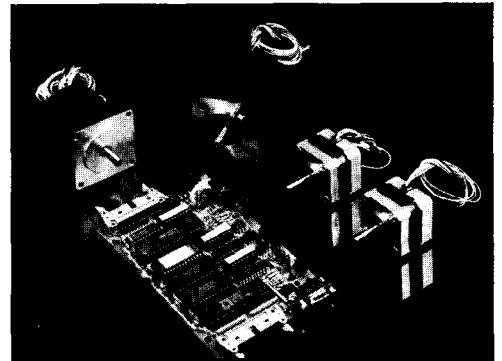
A complete stepper-motor-based motion-control system can be created with the High Stepper System from **CyberPak Co.** The HS-1 Multiaxis Indexer supports axis limit inputs and provides step and direction control signals simultaneously for up to four stepper motor drivers. It has a maximum step rate of **8000** steps per second and a total nonvolatile memory capacity of **128K** bytes. The HS-1 permits RS232 or RS-485 **multidrop**, which allows an expanded system to control up to 64 motors at once. It can be programmed as a stand-alone unit or can receive commands from a host PC or any other device supporting serial data communications.

The command set is composed of over 76 different commands including basic arithmetic; high-speed looping, calls, and branching; 208 variables; and general-purpose I/O functions addressing 48 off-board pins. The **HS-1** can be directly interfaced to user control panels involving matrix keypads, BCD switches, and LCD and VF displays. Isolated firmware drivers facilitate support for new interface devices. IBM PC software tools provide communications, program download and upload, custom ramp generation/installation, and so on.

The price of the HS-1 system starts from \$299.00 each in single quantities. Free technical application notes including hookup schematics to various brand stepper motor power amplifiers, matrix keypads, LCD displays, and thumbwheel switches are available on request.

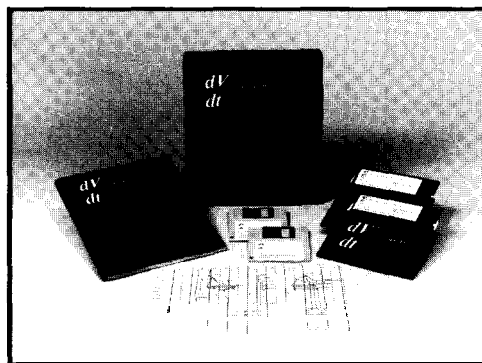
**CyberPak Co.**  
251 S. Frontage Road, Suite 23  
Burr Ridge, IL 60521  
(800) 328-3938 • Fax: (708) 654-4027

Reader Service 44217



## TIMING DIAGRAM ACCELERATOR

The analysis and optimization of digital circuit timing diagrams can be accomplished faster and more accurately with a new software product from Doctor Design Inc. **dV/dt** (the mathematical designation for acceleration) creates a unique approach to circuit design by integrating waveform sketching, circuit modeling, timing analysis, and verification into a single package. It allows design engineers to rapidly sketch and change timing diagrams, define time relationships between events, automatically reanalyze



design changes, and perform "what-if" calculations. Any circuit, including those with complex microprocessors or custom components, can be

easily analyzed, and bus timing requirements can be verified without captured schematics for processing. This "preprocessor" feature does not restrict the circuit **complexity** or require behavioral models, and does not require

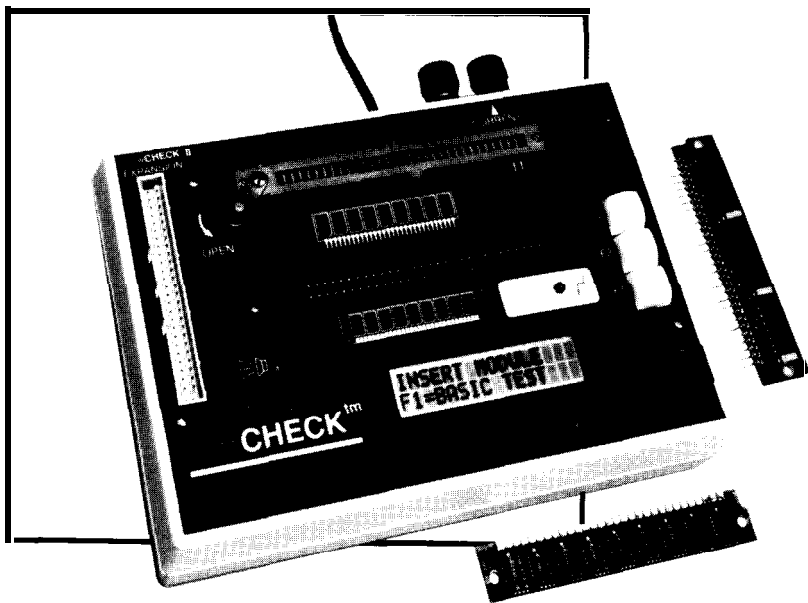
upgrading each time a **new** processor is released.

**dV/dt** generates clock signals automatically, performs common path and common part

analysis, defines propagation symbols, and verifies setups, holds, pulse widths, and cycle times. It provides automatic timing diagram documentation by outputting the finished product to laser and graphic printers. The program also provides a user-friendly interface with pull-down menus and mouse-based icon selection.

**dV/dt** lists for \$695 and runs on the Apple Macintosh, IBM PC/AT, IBM PS/2 models 20/30 and compatibles.

Doctor Design, Inc.  
54 15 Oberlin Drive  
San Diego, CA 92121  
(619) 457-4545  
Fax: (619) 457-1168  
Reader Service #218



## PORTABLE SIMM/SIP MEMORY MODULE TESTER

SIMCHECK, the first portable SIMM/SIP tester that tests the memory module as a complete unit, has been announced by Aristo Computers Inc. The unit tests all the standard SIMM and SIP memory modules with 8 or 9 bits of 64K, 256K, 1M, 4M, or 16M devices. It is a stand-alone tester with a high-speed 16-bit processor to control the proprietary test routines. Access time is measured down to 20 nanoseconds and a unique CHIP-HEAT mode warms the modules for temperature-dependent measurements.

A two-line alphanumeric LCD display provides the operating instructions and test results, including identification of bad chips, access time, and module type and size. All chips are tested simultaneously and an Auto Loop Test allows testing to be repeated, using changing data patterns and different algorithms, without user supervision. The test programs reside in a socketed EPROM to allow for future enhancements. Errors are traced to specific chips or module wiring problems.

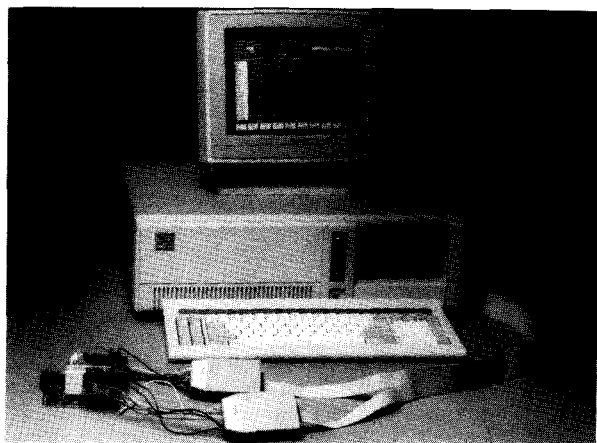
Zero-insertion-force sockets are used for both the SIMM and SIP modules, and full power protection using automatic current limiters and two programmable voltage sources is included. The unit measures 5" x 7" x 1.5" and weighs under two pounds.

SIMCHECK retails for \$995 and comes with a 30-day money back guarantee and one-year warranty. A full year of program upgrades is also included.

**Aristo Computers, Inc.**  
6700 SW 105th Avenue, Suite 307  
Beaverton, OR 97005  
(800) 327-4786  
Fax: (503) 626-6492

Reader Service X219

## PC-Based Logic Analyzers



### Sophisticated Logic Analysis a', Unsophisticated Prices

ID160 (50 MHz) for \$695

\*ID161 (100 MHz) for \$895

- 50 MHz or 100 MHz Sampling • 8K Trace Buffer • 32-channel Operation
- \*Multi-Level Triggering \*State Pass Counting
- \*Event Timer/Counter \*Performance Histograms \*Hardcopy Output
- \*Disassembles popular 8-bit micros • and much more!
- \*30 Day Money Back Guarantee



**INNOTECH DESIGN, INC.**  
6910 Oslo Circle, Suite 207  
Buena Park, CA 90621  
Tel: 714-522-1469 FAX: 714-527-1812

Reader Service #137

Announcing the

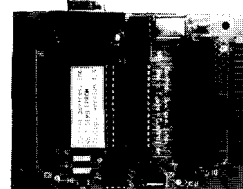
## WB-1 80C31 Industrial Controller

and Peripherals

Our expandable controllers get your projects up and running *fast!*

### Features include:

- Self-contained vertical stacking bus
- AT style DB9RS232C serial port
- Built-in watchdog timer
- Built-in power fail detection
- 8K battery backed RAM
- 8K EPROM
- Small 4" X 3" board size
- All address and data pins available on 80 pin bus
- 16 line decoder for memory mapping
- Surface mount technology utilized for glue chips
- Voltage regulator (+5 VDC)
- Reset button
- High quality dry film solder mask pc board
- Industrial temperature range available
- Diskette of assembly language support software included
- 24 hour support bulletin board (918)251-8031



**\$120<sup>00</sup>**

### Stacking bus peripherals include:

- @Breadboard with screw terminals and +5 VDC regulator **WB-1 BRD**
- LCD (20 X 4 char.), 16 button keypad interface & latched I/O **WB-1 LCD**
- Backplane with all power supplies and screw terminals for bus **WB-1 BPS**
- 82C55 parallel I/O board (total of 9 parallel ports) **WB-1PIO\***
- \*Multichannel A/D and D/A board (12 bit resolution) **WB-IAD\***
- Opto-isolated 4 20 mA current loops with surge protection **WB-1420\***

• Denotes cards available in the near future.

**CDI**  
A Product Development Company

2009 West Detroit Street  
Broken Arrow, Oklahoma 74012  
(918) 258-6068 Voice  
(918) 251-9851 FAX  
(918) 251-8031 BBS

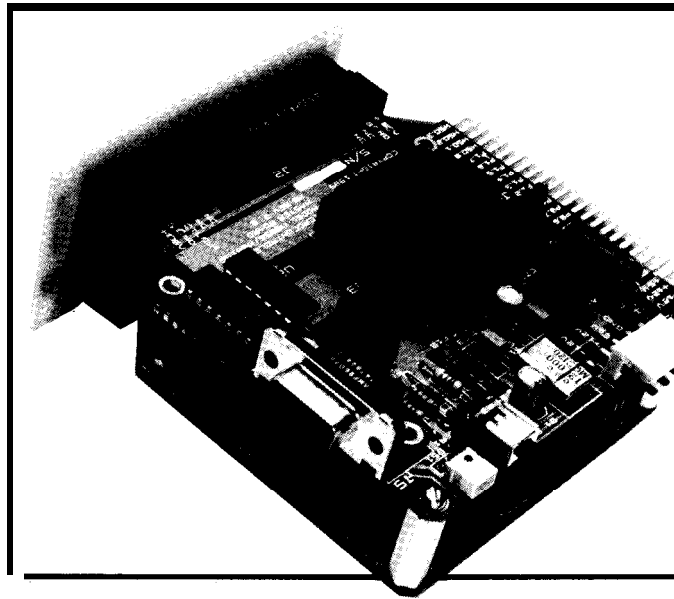
Reader Service #1

# NEWPRODUCTNEWSNEWPRODUCTNEWS

## MODULAR MICROCONTROLLER AND PROTOTYPE KIT

High-performance real-time control, with such applications as high-speed closed-loop motion control, midrange digital signal processing, and intelligent data acquisition, can be achieved with the MICON-196KC Modular Microcontroller and Prototype Kit from Micon Corp. The 3.5" by 3.5" unit features the Intel 80C196KC 16-bit embedded controller operating at 16 MHz, is software supported by an in-line monitor controlled via an IBM PC/XT/AT or compatible, and provides application-oriented tutorial programs. The MICON-196KC can also be used as an EPROM programmer for 87C196 parts and features a 64K SRAM/EPROM memory module with customized memory mapping.

The MICON-196KC includes eight ADC channels with sample and hold at a 9.8- $\mu$ s rate, four high-speed capture inputs with 1-microsecond resolution, six high-speed outputs for pulse and waveform



generation, one DMA channel, three pulse-width-modulated outputs (DAC), one full-duplex RS-232 serial port, and five 8-bit I/O ports.

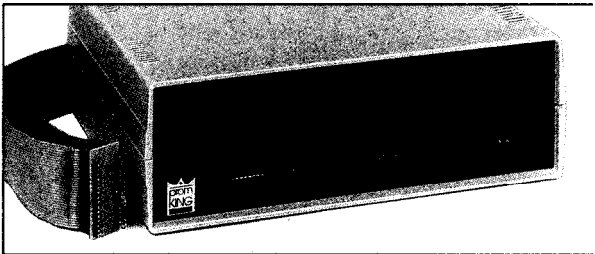
The MICON-196KC consists of CPU, memory, bus, and prototype modules, two monitor EPROMs, two PALs for memory configuration, a PC interface,

5.25" diskette, and user's manual with applications-oriented tutorial programs. The system sells for \$279.00. A power and communication hardware kit, the PSMICON-2, consisting of a compact power supply and cables, sells for \$89.00; and an L-8096-AL 8096 assembly language tutorial sells for \$78.00.

**Micon Corp.**  
5270 Elvira Road,  
Bldg. 104  
Woodland Hills, CA 91364  
(818) 348-4992  
Fax: (818) 348-0960

Reader Service X220

## STOMP OUT EPROM MADNESS



The PROM KING emulates EPROMS, saving both time and money during your development cycle. Programmable in seconds via your PC printer port or any computer RS232 port, it can emulate most 27xxx devices.

- 8K-8M bit devices
- High speed download:
  - Universal RS232
  - PC printer port
- Menu driven software
- Battery backup
- 8-256 bit downloads
- Easily expandable:
  - 4 EPROMS per unit
  - Up to 8 units
- Also programs like a real EPROM

\$599 for 150nS units with 256K bits. Ask for pricing of other options,

Made in USA by

**TRAXEL LABS INC.**  
BOX 239 • RONKONKOMA, NY • 11779  
516-737-5147 FAX • 516-737-0349

QUALITY PARTS • DISCOUNT PRICES • FAST SHIPPING

## ALL ELECTRONICS CORP.

P.O. Box 567 • Van Nuys, CA • 91408

### 12 VDC GEAR MOTOR

Soho# GEL 35-DH-21080-10Y

Powerful little gearhead motor.

40 RPM @

12 Vdc. (no load).

32 RPM with load. Operates at lower voltages with reduced speed and torque. 6.3 pound inches torque. Stall: 27 pound inches. 3.1" long X 1.375" diameter. Shaft: 6.167" dia. **CAT# MOTG-14**

\$11.50 each • 10 for \$100.00

### OPTO-ISOLATORS

Clairex# CLM-6000

LED-photoconductor

Isolator. Off resistance: 500 ohms. On resistance: 500K ohms. 2000 volt isolation. Forward voltage: 2 Vdc. **CAT# CLM-6000**

\$2.50 each • 10 for \$22.00

Sigma# 301T1-12B1. Signal applied to the input is coupled by means of light to isolated photo conductive cell. High reliability switching. 12 Vdc.

**CAT# OP-301** \$1.50

### OPTO SENSOR

U shaped package with mounting ears. 1/8" opening.

3/4" mounting ears. **CAT# OSU-6**

50c each • 10 for \$4.50 • 100 for \$40.00

### INSTRUMENT ENCLOSURES

Molded ABS instrument enclosures. Matching front and rear panel. Integrated PC board standoffs and two sets of

vertical mounting slots for front and rear sub panel PC boards. All enclosures are 6" wide X 6 1/4" deep. Available in black, ivory, blue, and beige. Specify color.

FRONT & REAR PANEL HEIGHT

2 1/4" CAT# MB-A \$7.50 each 10 for \$65.50

2 5/8" CAT# ME-B \$7.75 each 10 for \$67.50

3" CAT# MB-C \$8.00 each 10 for \$70.00

### PHOTOFLASH CAP.

Rubicon CE 210 MFD 330 V

3.79" dia. X 1.1" high. New.

prepped with 1.4" black and red wire leads soldered to the terminals.

**CAT# PPC-210** \$2.50 each

10 for \$22.50 • 100 for \$200.00

### STEPPING MOTOR

Airpax# CB2711-M1

17 Vdc 23.25 ohm

dual coil P.M. motor.

7.5 degrees per step.

2.25" dia. X .91" thick.

3.25" dia. shaft. 6 wire leads.

**CAT# SMT-6** \$6.00 each • 10 for \$50.00

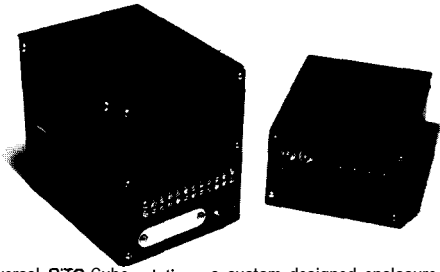
**ORDER TOLL FREE 1-800-826-5432**

**FAX (818) 781-2653** Terms: Phone orders must be charged to Visa, MasterCard, or Discover. Minimum order \$10.00 • CA residents add sales tax • 48 Continental States Add \$3.50 for shipping/ handling - All Others

Including AK and HI pay full shipping • Quantities limited • No C.O.D.

**CALL TOLL FREE FOR A FREE 60 PAGE CATALOG!**

## Universal RiTC Cube



- Universal RiTC Cube **solution** - a custom designed enclosure for Micromint RTC-based applications. Supports Micromint PS05S P.S.
- Dimensioned for the Micromint RTC family, 5" x 6" footprint.
- Constructed from 15 Gage brushed black-anodized aluminum.
- Convertible box : 7 board or 3 board stack **capacity**.
- industry standard connector cutouts on faceplates.
- 4-40 machine screws and 4-40 PEM nuts for assembly
- Optional 0.531" standoffs available for securing 'RTC stack'.

**RiTC-CUBE:** \$99.95

Headlight Kit - 7 LED's + driver, power & reset **switches** all on a single board (RTC-SIR IR LED supported). Mounts in Universal RiTC Cube faceplate

**RiTC-HEADLIGHT:** \$29.95

Communication Kit - A pair of RJ12 sockets configurable for MC-NET or dual serial **ports** on a board. Mounts in Universal RiTC Cube faceplate.

**RiTC-COMKIT:** \$29.95

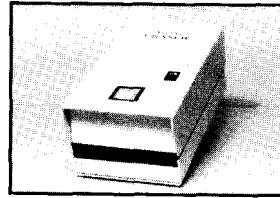


Integrated Vessel Information Corporation  
871 Via Alondra, Unit 805  
Camarillo, California 93010  
(505) 3595870

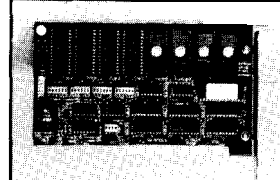
## ELECTRONICS

1 2 3  
A DIVISION OF MING E&P, INC

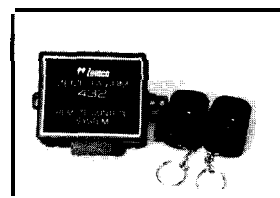
1. Exclusive items at good price.
2. Unique items at better price.
3. Popular items at the best price.



**5 SECOND EPROM ERASER**  
Revolutionary product  
super energy output  
Saves time & money  
The most desired product  
Patented design  
MING IEB9088 \$249.99



**ROM/ SRAM DISK CARD**  
For diskless PC station  
Load DOS & file instantly  
Battery back-up for **SRAM**  
Watch-dog time; rebooting  
**RD512 (512KB, 0KB)** \$179.99  
**RD1024 (1024KB, 0KB)** \$199.99



**RF REMOTE CONTROL SYSTEM**  
19683 digital coding  
2 tiny transmitters  
Dry contact relay output  
**ON/OFF** confirming signal  
FCC approved  
**ZEMCO SA432** \$49.99

**1-(800)-669-4406**  
TOLL FREE ORDER LINE

977 S. Meridian Ave., Alhambra, CA 91803  
Tel: (818) 281-4066 Fax: (818) 576-8748  
VISA & MASTER CARD ACCEPTED

Reader Service #138

Reader Service #13

# Computer Professional's Legal Toolkit

For Programmers, VARs, Consultants, Retailers, Developers,  
Hardware/Peripheral Manufacturers - Any Computer Entrepreneurs!

**Problem:** One of the world's largest chipmakers forfeited all legal protection for a revolutionary new product due to a SIMPLE copyright law mistake!

**Result:** Over \$200,000,000.00 in lost revenue!

**Solution:** THIS PROGRAM!

**Only \$159.95**

Order NOW!  
Call 24 Hours!  
Most Cards Accepted!

**1-800-648-1300**

ONE LAWSUIT CAN RUIN YOUR WHOLE DAY!™

Full Annotated Text of Laws,  
Court Cases and Government  
Regulations +  
Mini-Seminars +  
Legal Document Generator Using  
A.I. with More Than 40 Forms  
That Are Valid In All 50 States!

U.S. Copyrights  
International Copyrights  
Uniform Trade Secret Law  
Patent Law  
Trademark Law  
Computer Crimes Law, Federal  
Computer Crimes Law, all 50 States  
Warranty Disclaimers  
Military Computer Acquisition Regulations  
Legal Liabilities of Computer Professionals  
Beta Testing Law  
BBS Laws  
Employer/Employee Relations  
Computer and Software Leasing  
Credit and Collections  
Contract Bidding  
Semiconductor (Mask) Work Protection  
Arbitration  
I.R.S. Issues Relating to Computers  
Patenting vs. Copywriting

Written by a computer-literate attorney

**PRIVATE BBS • COMPUTER EMPLOYMENT EXCHANGE. UPDATES**

Minimum System Requirements: MS-DOS XT, AT, 386 Compatible, 512k RAM, Hard Drive

R. Fringe Publishers, Inc.  
Box 796, Casselberry, FL 32707  
© 1990 Kraft & Byron

# FEATURE ARTICLE

*Chris Clarcia*

**T**he increased use of imaging systems in the world of microcomputers is primarily due to emerging low-cost instrumentation Cameras, scanners, high-resolution monitors, mass storage devices, video printers, image capture-digitize-display boards, as well as advanced packages in image synthesis and processing software, have become generally available at affordable prices. As a result, there has been growing popular support for a whole array of imaging applications. The use of these imaging technologies also implies the use of some very sophisticated data handling utilities if you need to transfer the 'imaged' data at reasonable rates across some network or modem connection.

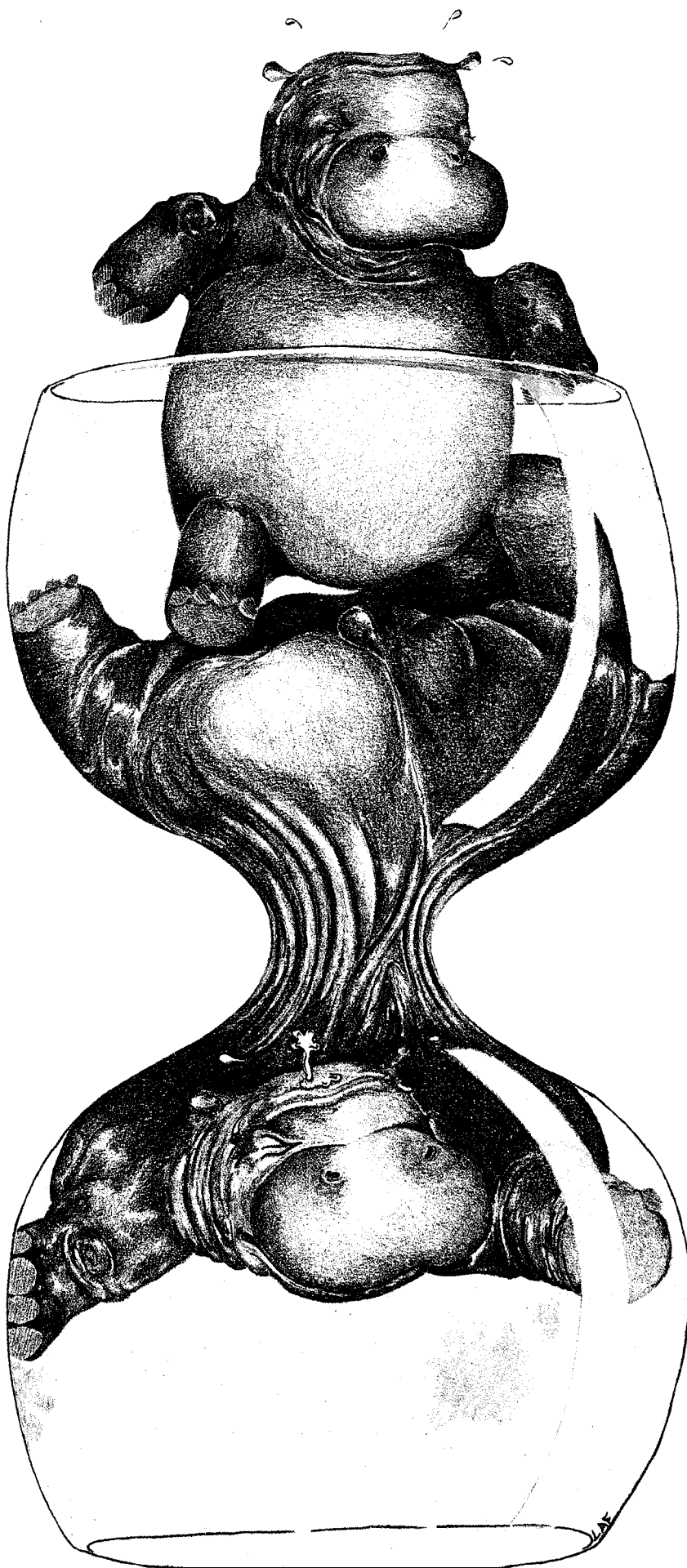


Illustration by Lisa Ann Ferry

# Image Compression for High-Speed Network Transmission

I know! I've been especially aware of this problem since it takes me over 15 minutes to transfer a 512 x 512-byte image over my modem at 2400 bps. The old saying, "a picture is worth a 1000 words," is underestimating the truth.

With the growth of imaging applications in networked PC systems, techniques that enhance "image data" transfer have been high on the R&D list. Image data compression has become an integral part of imaging systems technologies. As a result, there is intensive ongoing research in such areas as digital mapping, document archival and retrieval, electronic publishing, engineering drawing, image communication, medical imaging, cataloging, picture ID systems, point-of-sale systems, prepress imaging, remote surveillance, teleconferencing, telemetry, teleradiology, image synthesis, animation, and artificial vision within AI and robotics. Each of these fields has different image compression requirements.

Of course, the choice of compression mode depends highly on the type of image data. Using a data compression utility that is truly designed to be universal is highly desirable. It should be able to handle different types of image data which support a wide variety of applications. To that end, I'll describe a simple programmable image compression system which can compress and expand single-frame computer-generated graphics images (monochrome or color) as well as scanned documents and video images. It is programmable by the general user who can't afford the fancy "hardware" compression card. My method uses a very simple, but universally applied compression technique which is based on Fourier analysis methods. Through the application of a Fourier transform,

an image is decomposed into its frequency components. These components are then selectively chosen (banded) according to their contribution to the visual content of the original image. All components that have little effect on the image are eliminated in order to effectively reduce the size of the information set necessary for a reasonable reconstruction. These "necessary" components are then quantized into integer-step (level) values for encoding into a binary-word data stream. The data stream is then compressed by the elimination of redundancy and multiple spatially adjacent zeros for transmission, and reconstructed by reversing the procedure.

## IMAGE DATA TYPES

Different digital image data types are generally categorized and defined in terms of their color content; whether the image is monochrome or color and if it is colorized, whether the color is classified as true or mapped color.

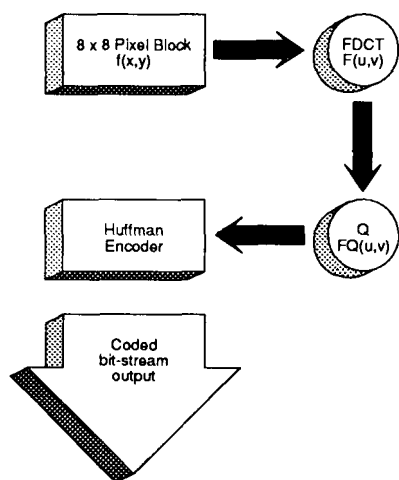


Figure 1 - The DCT coding technique is most useful on images with a high degree of local pixel correlation.

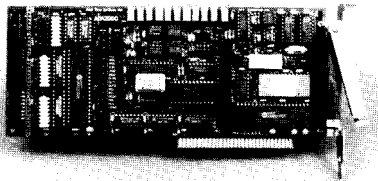
For monochrome images, the number of bits per pixel fully defines the image value, or gray level; with typical systems employing one of three gray level schemes, either a single binary image composed of only two gray levels, an 8-bit system composed of 256 individual gray levels, or a 12-bit, 4096 gray level format. With the addition of color, the number of bits necessary to encode information increases. For example, if we consider an RGB true color scheme, we need to define an n-number of bits for each of the three individual components per color pixel. In a mapped color system, each pixel value is designed to represent an index into a larger palette, with a small number of key colors or (RGB triplets) being selected from the palette for a given application. Here, mapping is implemented through the use of a look-up table, with the resulting color value containing so many index bits and three n-bit color components. As a result, true colors are typically represented by a 3-byte word using 8 bits for 224 different combinations (16,777,216 colors) or 5-bit words representing 215 combinations (32,768 colors); while mapped color systems usually employ an S-bit index with an 8- or 6-bit 3-word length palette, resulting in 256 colors from a palette of either 224 or 218 (VGA) colors respectively.

As a result of these high densities in image pixel data bits, the major decision lies not so much in what type of data format is used, but whether one implements a lossless or controlled quality compression technique. A lossless method of compression implies that the reconstructed image will be digitally identical to the original. In controlled quality compression, the image is reconstructed with reasonable image quality but using less in-



# PC Bus Data Acquisition and Control

Quality U.S.-manufactured cards and software for single user, OEM, or embedded applications.



## ADA2000 - \$589

8-Channel, differential, 12-bit, 20 $\mu$ s A/D  
Programmable gains of 2, 4, 8, & 16  
Three 5-MHz timer/counters  
Two 12-bit D/A outputs  
40 Digital I/O lines  
120 signals through a single slot!  
Dedicated ground for each analog signal

Real Time Devices, Inc. designs and manufactures a broad line of cost-effective industrial/scientific interface cards and software for the PC/XT/AT bus. Our commitment is to offer only high-quality U.S.-designed and manufactured interfaces with emphasis on signal quality and ease of use for OEM applications. All our cards are backed by a one-year warranty, and 30-day NO-RISK return policy. Call today to request your free catalog and discuss your application requirement!

**AD1000** X-channel 12-bit 20  $\mu$ s A/D; sample & hold; three 5-MHz timer/counters; 24 TTL digital I/O lines. .... \$325  
**AD2000** X-channel differential 20  $\mu$ s A/D; sample & hold; three 5-MHz timer/counters; 2,4,8,16 prog. gain; 16 digital I/O .. \$495  
**AD100** 1-channel single-ended 12-bit integrating A/D; 1,10,100 prog.gain.... \$159  
**AD200** 4-channel 12-bit 125  $\mu$ s A/D; three 5-MHz timer/counters; resistor-configurable gains; 24 digital I/O lines. .... \$259  
**AD500** X-channel 12-bit integrating A/D; programmable gains of 1, 10, & 100. Extremely stable, accurate & sensitive. .... \$259  
**ADA100** Single-channel, differential input, 12-bit integrating A/D; 8-bit D/A output; programmable gains of 1, 10, & 100. Plus 10 digital I/O lines. .... \$215  
**ADA300** X-channel 8-bit 25  $\mu$ s A/D; single 8-bit D/A; 24 TTL digital I/O lines .... \$259  
**DA600/DA700** Fast-settling 2/4/6/8 -channel 12-bit D/A; double buffered .. \$207/1495  
**DG24/96** 24/48/72/96-line TTL compatible digital I/O cards; NMOS 8255-based. Opt. buffers and pull-up resistors .. \$110/1274  
**TC24** Five 5-MHz timer/counters; uses powerful AM9513 chip; 24 digital I/O lines from NMOS 8255 PPI chip ..... \$218  
**ATLANTIS** High-performance data acquisition software; foreground/background operation; maximum 25-KHz rate; supports hard disk streaming; pull-down windows.. \$250

Custom/OEM designs on request!

Real Time Devices, Inc.

820 N. University  
P.O. Box 906  
State College, PA 16804

Phone: 814/234-8087

FAX: 814/234-6864

Reader Service #168

formation during transfer. That is, the compression ratio is enhanced at the cost of image reproduction quality. The compression ration (CR) is defined as the ratio of  $b/c$ , where  $b$  is the number of bits per pixel in the original image and  $c$  represents the bits per pixel in the compressed image. For example, if my 512 x 512 by 8-bit byte image is made of 161,319 bits, in compressed form, ( $b = 8$  bits/pixel and  $c = 161319/(512 \times 512) = 0.61538$  bits/pixel) then my compression ration is  $8/0.61538 = 13.0$ . I have made my data transfer 13 times faster using compression. Of course, in the real world I need to include time spent for compression/decompression execution, so I will actually realize a total transmission factor on the order of 5 to 10, depending on the form of the utilities. Software implementation is the slowest compression technique, but using a hardware compression board (such as one structured around the Zoran ZR34161 vector signal processor and an Intel 80286 CPU), extremely high factors are possible.

To give you one example of these "hardware implemented" compression-transfer capabilities, consider an image made up of 512 x 480 true color,

RGB 8-bit word pixels. An achievable compression ratio for such an image is 24:1[1]. Using Zoran's card, this image can be compressed in 4.7 seconds and decompressed in 2.35 seconds, resulting in a total of 7.05 seconds needed for the compression/decompression cycles. Now assume that our network transfer information at 9600 bps and the total image contains 512 x 480 x 24 bits. As such it will take 614.4 seconds (or 10.24 minutes) for the uncompressed version to be transferred. Using the processor-based system with a compression ratio of 24, this same file transfer will only take 25.6 + 7.05 seconds (or 0.54 minutes). This doesn't include the time for loading the image into and out of the frame buffer for line transmission.

## A PROGRAMMABLE DCT COMPRESSION UTILITY

Reducing the amount of image data stored or transmitted greatly reduces the disk capacity or channel bandwidth required in a system. Unfortunately, as the quantity of the information used to represent an image is decreased, so is the subjective quality of the image. Most **compres-**

```

c
c set coefficients c(u),c(v)
c
do i=0,7          ! N=8 dimension of block, N-1=7
  if (i.eq.0) then
    C(i)=0.7071068 ! 1/SQRT(2)
  else
    C(i)=1.0
  endif
enddo

c
c do 2-D DCT for N=8
c
PI=3.141593
do u=0,7
  do v=0,7
    F(u,v)=0.0
    do x=0,7
      do y=0,7
        du = ((2*x)+1) * PI * u / 16
        dv = ((2*y)+1) * PI * v / 16
        F(u,v) = F(u,v) + f(x,y) * cos(du) * cos(dv)
      enddo
    enddo
    F(u,v) = (1/4) * F(u,v) * C(u) * C(v)
  enddo
enddo
enddo
enddo

```

listing 1 - One method for obtaining a 2-D DCT is by using the forward application of the two-dimensional DCT algorithm (FDCT).

sion algorithms must therefore make a compromise between the transmission rate (compression ratio) and their ultimate image quality. This rate-versus-quality tradeoff has led to the use of a popular method of image compression called the Discrete Cosine Transform (DCT) technique, which minimizes this compromise. The DCT (and its inverse, IDCT) is widely used for compressing motion and single-frame coding and has been proposed as the international video-telephony

The DCT coding technique is most useful on images with a high degree of local pixel correlation. The technique involves the partitioning of an image into 8 x 8 blocks (defined here as  $f(x,y)$ ), with each block being acted upon by a two-dimensional DCT. The output of the DCT procedure is then passed through a quantizer, an algorithm chosen to trade off perceived quality against bit rate and then passed to the Huffman encoder for coded bit-stream construction, as shown in Figure 1.

```

PI=3.141593
do k=0,N-1      ! the dimension of the N x N block
  if (k.eq.0) then
    c = 0.7071068
  else
    c = 1
  endif
  do m=0,N-1    ! for each sample input x(m)
    d = ((2*m)+1) * PI * K / (2*N)
    X(k) = X(k) + x(m) * cos(d)
  enddo
  X(k) = X(k) * SQRT(2/N) * C
enddo

```

**listing 2**—An alternate method for obtaining a 2-D DCT is to perform the 1-D DCT eight times in each dimension. Both forward and inverse (shown below) transforms are used. Here,  $x(m)$  represents the input samples and  $X(k)$  is the resulting 1-D DCT output, with  $N=8$  for most applications.

```

PI=3.141593
do m=0,N-1      ! the dimension of the N x N block
  do k=0,N-1    ! for each sample input x(m)
    if (k.eq.0) then
      C = 0.7071068
    else
      C = 1
    endif
    d = ((2*m)+1) * PI * K / (2*N)
    x(m) = x(m) + X(k) * C * cos(d)
  enddo
  x(m) = x(m) * SQRT(2/N)
enddo

```

**listing 3**—The inverse 1-D DCT is used with the forward 1-D DCT (above) to obtain a 2-D DCT. Here,  $x(m)$  represents the output pixel values and  $X(k)$  is the input.

standard. DCT compression also has the advantage of being cheaper to implement than most other techniques, such as subband coding [2], since it renders an image into frequency components, a process that can be achieved by a single pipelined operation. For images of medium resolution, say 350 x 250 pixels, a disk bandwidth of about 1.0 megabits/second produces a good reconstructed image. This corresponds to about 2.5 minutes of full color motion video from a 20-megabyte Winchester disk and over twenty times that figure from current technology CD ROMs.

The output of a 2-D DCT operation,  $F(u,v)$ , can be obtained using two different methods. The first is achieved by the forward application of the two-dimensional DCT algorithm (FDCT) shown in Listing 1, with  $F(u,v)$  being the output of the FDCT and  $f(x,y)$  being the input 8 x 8 pixel block. For an inverse 2-D DCT (IDCT), apply the following equation:

$$f(x,y) = \frac{1}{4} \sum_{u=0}^7 \sum_{v=0}^7 C(u) C(v) F(u,v) \cos\left(\frac{2x+1}{16} u \pi\right) \cos\left(\frac{2y+1}{16} v \pi\right)$$

## FAST TEST DRAM DIPs - SIMMs - SIPs



## RAMSTAR

1ns. RESOLUTION

### ACCESS SPEED VERIFICATION

80 ns. thru 180 ns. (Std.)	\$249.00
45 ns. thru 110 ns. (Fast)	\$349.00
4M Option	Add \$ 89.00

### AUTO-LOOP

Continuous Test 6.25 MBits/sec.

### ADAPTERS:

<b>SIMM/SIP ADAPTER</b>	\$189.00
Tests 64K, 256K, 1 M & 4M Devices 8 or 9 Bit versions	

### NEW

<b>SIMSTAR ADAPTER</b>	\$431.00
Tests All Bits Simultaneously 64K, 256K, 1 M & 4M Devices 8 Bit, 9 Bit & PS/2 SIMMs or SIPs <b>Production Test Model</b>	
	\$531.00

<b>4 X ADAPTER</b>	\$ 89.00
Tests 64K & 256K By 4 Bit Devices	

<b>AC ADAPTER</b>	\$ 18.00
Regulated +5V @ 1 Amp.	

**FREE RAMFACTS  
DRAM NEWSLETTER**

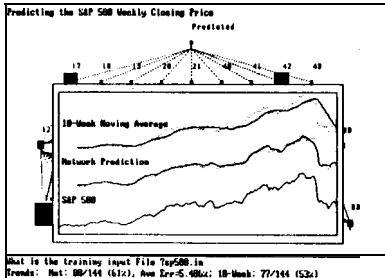
## 1-800-RAMSTAR

COMPUTER DOCTORS  
9204-B Baltimore Boulevard  
College Park, Maryland 20740

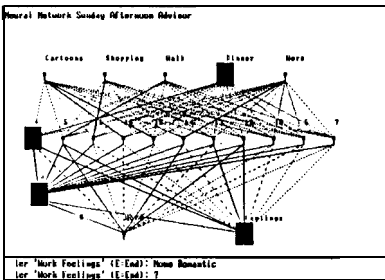
MADE IN U.S.A. PATENT PENDING

Reactive Services #117

# Introducing NeuralWorks™ EXPLORER



## Stock Market Forecasting



## Expert Systems

NeuralWare, Inc. presents NeuralWorks Explorer, a neural network tutorial that provides the novice user with a method of learning neural network theory as well as an environment in which to build practical applications. Available on both the MAC and PC. Price \$199.

The NeuralWorks product line is currently used in:

- Oil Exploration
- Medical Diagnostics
- Industrial Inspection
- Credit Approval
- Process Control
- Insurance Underwriting
- Economic Modeling
- Noise Filtering
- Signal Processing
- Fluid Detection
- Bankruptcy Prediction
- Targeted Marketing



**NeuralWare, Inc.**

Penn Center West, Bldg. IV, Suite 227  
Pittsburgh, Pennsylvania 15276

**412-787-8222**

Reader Service #160

```

do u=0,N
  do v=0,N
    if (F(u,v).ge.T) then
      FQ(u,v) = (F(u,v)-T) / g + 1
    else if (F(u,v).gt.-T .and. F(u,v).lt.T) then
      FQ(u,v) = 0
    else if (F(u,v).le.-T) then
      FQ(u,v) = (F(u,v)-T) / g - 1
    endif
  enddo
enddo
  
```

Figure 4—A simple way of quantizing the output of the DCT can be realized through the use of the quantization algorithm shown here.

The second procedure for obtaining a 2-D DCT performs the 2-D DCT by using the 1-D DCT eight times in each dimension according to the algorithms from Listing 2 and Listing 3. Both  $m$  and  $k$  represent indices of line DCTs along a row or column of the block,  $f(x,y)$  and  $F(u,v)$ .

## THE QUANTIZATION PROCESS

Next, each of the 64 FDCT output coefficients of  $F(u,v)$  are quantized by a uniform quantizer. Usually, the FDCT block of spectral components forms a sparse set with the majority of

the energy concentrated in the low-frequency corner of the block. Special care must therefore be taken with the choice of a frequency threshold and the quantization step-size if we are to maintain reconstruction quality. In principle, a quantization matrix is constructed to be the perceptual discriminator for the visualization contribution of the cosine basis function of frequency  $(u,v)$ , for the intended color system, display device, and viewing distance; with multiple  $Q$  matrices being created for different color components since their perceptual contribution varies. A simpler way of quan-

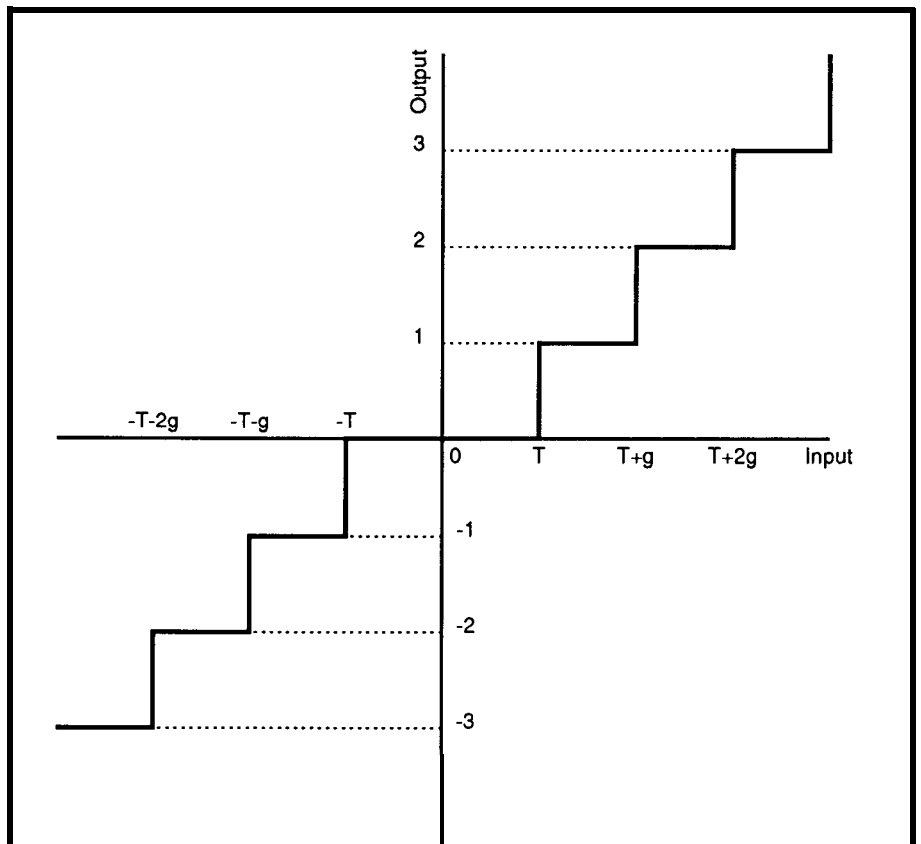


Figure 2—In the quantization algorithm shown above,  $g$  can be varied depending on the desired transmission bandwidth. Here, the quantizer function is represented for the case  $g \neq T$ .

tizing the output of the DCT can be realized through the use of the quantization algorithm in Listing 4 (for a single threshold and step size), where  $FQ(u,v)$  is the step-wise quantized data,  $F(u,v)$  is the output of the DCT function,  $T$  is the threshold at which

term and 63 zero terms, for a compression ratio of 64:1. In addition, many frequency components are small and contribute little to the overall image. They are below the visual threshold and are truncated in the quantization process. These now zero components represent the majority of the available image compression. For higher compression ratios the quantization step size and threshold can be raised.

For proper reconstruction, prior to the original bit compression, the "DC" coefficient,  $FQ(0,0)$  is usually treated separately from the other 63  $FQ(u,v)$ -quantized "AC" coefficients (within the matrix, rows represent increasing horizontal frequency, columns represent increasing vertical frequency, and  $FQ(0,0)$  is the DC term). The Quantized DC term from block  $i$  is differentially encoded with respect to the DC

term from the previous block  $i-1$ :  $d/dt DC(i) = DC(i) - DC(i-1)$ . Since 11 bits are sufficient to represent any quantized DCT coefficient (either AC or DC), 12 bits are sufficient to represent all possible differential terms.

Prior to Huffman Coding, the 63 quantized AC coefficients are ordered from their 2-D matrix into a 1-D sequence, in a "zig-zag" manner starting at  $FQ(0,1)$  (i.e., 1 = (0,1), 2 = (1,0), 3 = (2,0), 4 = (1,1), 5 = (0,2), 6 = (0,3), 7 = (1,2), 8 = (2,1), etc.)

## THE HUFFMAN ENCODER

The Huffman technique uses variable-length codes to represent the values of our quantized DCT string sequence (of  $n$  bytes), according to the frequency of occurrence of the value as shown in Table 1.

If most of the nonzero  $FQ(u,v)$  coefficients have the value 1 or 2, then coding them can approach two bits per coefficient (one bit for the value plus one bit for the sign). Since the probability of a zero is even higher than a one and because zeros are more likely to occur in the high-order coefficients, a simple code can be created to represent multiple spatially adjacent zeros. Here, additional enhancement of the compression ratio is possible depending on the truncation and ordering scheme chosen to optimize adjacent zero specification [4,5].

## HARDWARE IMPLEMENTATION

Use of the DCT utility for compression purposes has become very popular. Since its introduction as an industry standard, many DCT-specific processors have been created and several PC image compression boards have been designed.

One such DCT processor is the IMS A121, shown in Figure 3. It performs DCT and IDCT by the classical matrix multiplication method using a distributed arithmetic architecture which is ideal for cheap dedicated arrays with fixed coefficients stored in

Coefficient Value	Huffman Code	Number of Bits
1	1	1
2	011	3
3	01000	5
4	01010	5
•	•	•
zeros	00	2

with the number of bits  $b$  required to represent a value of probability  $P$  being,

$$b = -\text{LOG}_2 P$$

where  $b$  is rounded to the next higher integer.

Table 1—The Huffman technique uses variable-length codes to represent the values of the quantized DCT string sequence.

the output is forced to zero, and  $g$  is the quantization step size. Note that  $g$  can be varied depending on the desired transmission bandwidth. Figure 2 is a representation of the quantizer function for the case  $g \neq T$ .

Since most images do not contain all frequencies, as a white noise image would, many of the values of  $FQ(u,v)$  will be set to zero. This represents the redundancy within an image and its available compression. For example, a pure gray image would have one DC

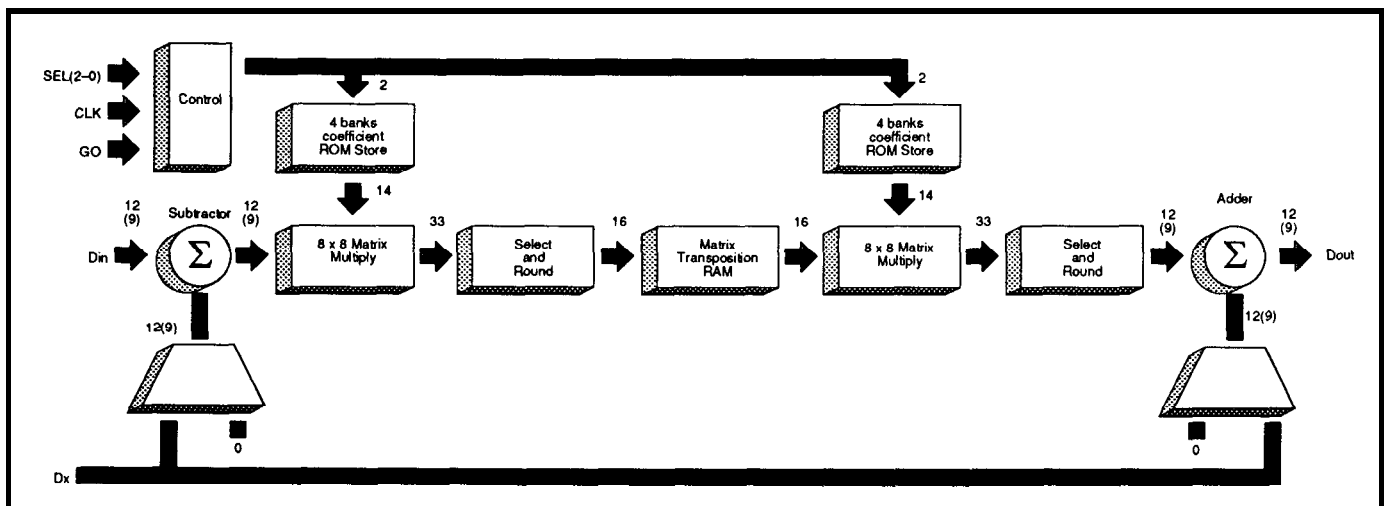


Figure 3—The IMS A121 DCT processor performs DCT and IDCT operations on a single chip.

preprogrammed ROM. The device is manufactured in **1.2-micron** CMOS and contains 185,000 transistors. The most current specification uses a **20-MHz** clock, which corresponds to a processing rate of 320 MOPS. As such, it can calculate a transform in 3.2 microseconds.

A very effective image compression card for your PC is currently available from Zoran Corporation of Santa Clara, California, called the Image Engineering Board (IEB). It is a three-bus dual-processor system (ZR34161 vector signal processor and Intel 80286 CPU) optimized for high-performance and Fourier domain processing, and operates in an AT environment. The memory resources support processing of a 512 x 480 RGB image, and can be flexibly allocated as data, image, or program memory. A 2K DPR supports simultaneous processor operations (see Figure 4).

For a 512 x 480 RGB image, the IEB has demonstrated a compression time of 4.7 seconds at a compression ratio of **24:1**. This time includes DCT, encoding, and image transfer from AT frame buffer to IEB and back. The measured decompression time for the IEB is 2.35 seconds. For more details on this board, I refer you to a paper by I. Livny and D. Seltz [1], "Vector Signal Processor-Based Image Processing Applications," presented this last October at the Boston Electronic Imaging '89 Conference.

#### AND ON TO THE REAL WORLD

I don't know about you, but I'm an impatient man. I dislike staring at my CRT and waiting for the end of an image file transfer. That's what got me into the compression business in the first place. Since I prefer not to purchase fancy "hardware," I developed the need and opportunity for looking into how I could implement my own image compression/decompression utility on my **386/20**. I built a simple FORTRAN code based on the DCT technique described above. It's not bad. It speeds up my modem transfer time by about a factor of eight; so, I only hang around for about two minutes instead of 15-20 (for an **8-bit**

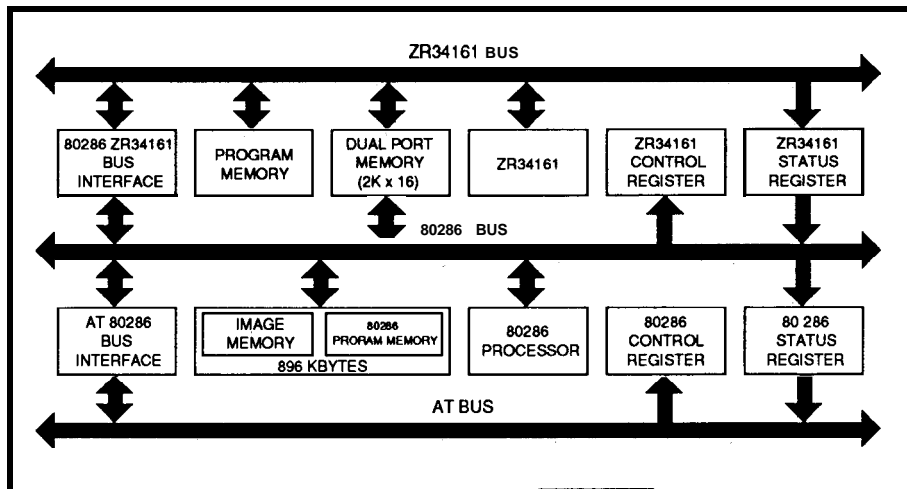
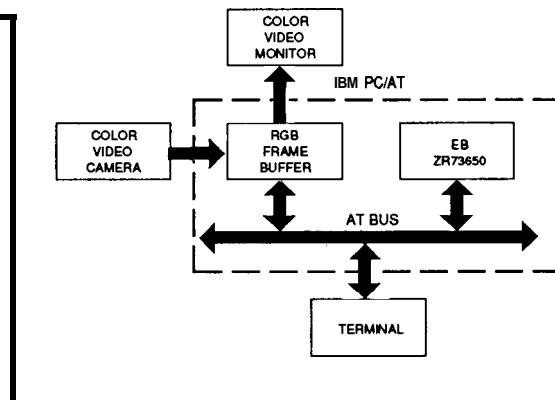


Figure 4—The Zoran Image Engineering Board is a three-bus &al-processor system (ZR34161) vector signal processor and Intel 80286 CPU) optimized for high-performance and Fourier domain processing.



# PROGRAMS > 64K?

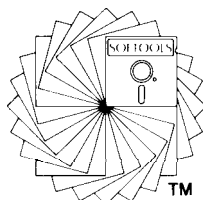
**Auto-MMU Support Is The Answer.**

**SASM-Advanced Macro Cross Assembler**  
**SLINK-Advanced Linker**

**Softools, Inc.** introduces a relocating macro assembler and linker package that offers many features for the **embedded programmer** at an affordable price. It supports the 64180, 280, 8085, and 2280 processors.

**SASM** also supports the **64180 MMU** for automatic control of programs larger than 64K by making "long" calls into segments not mapped into the address space. It also includes many pseudo-opcodes for close compatibility with other assemblers. **SASM** accepts expressions that use operators common with other assemblers as well as C operator equivalents. **SLINK** is able to resolve any expression if **SASM** is unable to obtain a result. **SASM** includes a built-in **MAKE** facility which supports dependency file checks. It allows you to use one source file to generate a multi-module library file. In addition, **SASM** generates full source-level debugging information for each source file including the source name, include files, line numbers, public symbols, and local symbols.

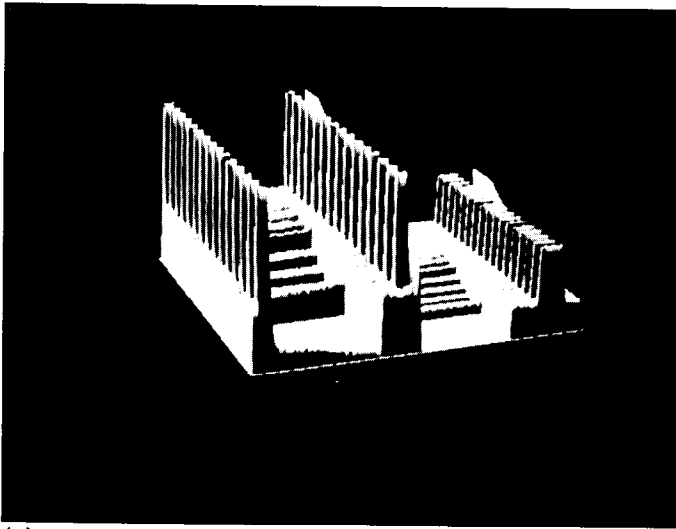
**SLINK** output is compatible with In-Circuit Emulator (ICE) source-level debugging, and also generates binary or Intel HEX files and has the ability to divide output into multiple ROM image files. It supports named segments which may be up to 64K in length each, and may be linked to reside at one physical address and executed at another. Any banked or MMU controlled program requires this feature to locate code effectively. **SLINK** also allows the exclusion of physical address ranges in order to leave holes in the output file.



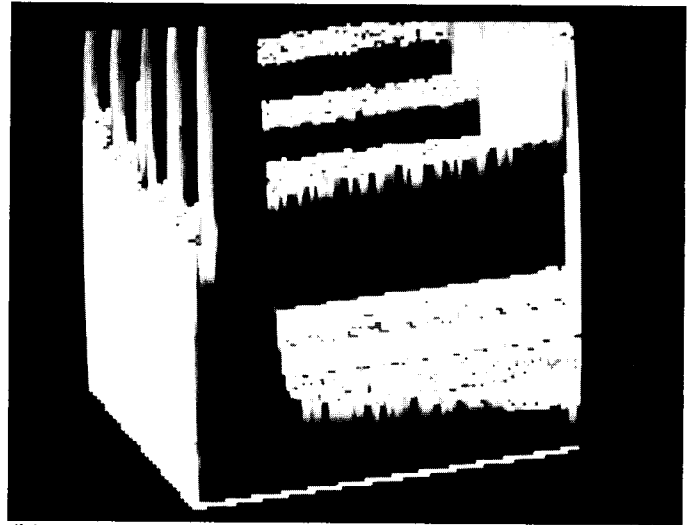
**SOFTOOLS INC.**

8770 Manahan Drive  
Ellicott City, MD 21043  
301-750-3733

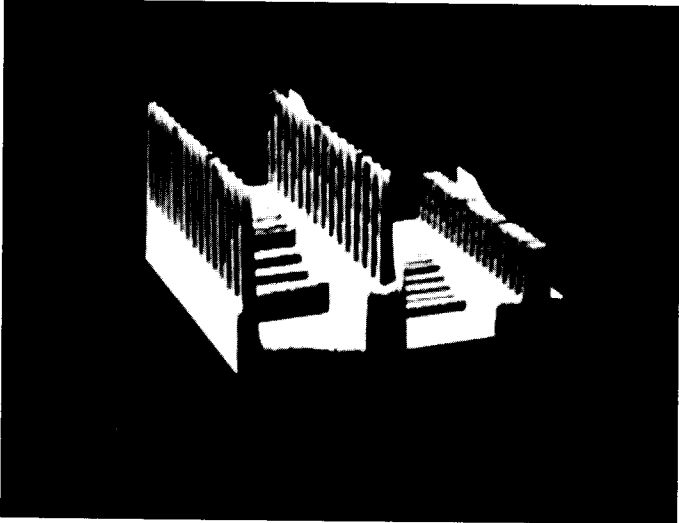
**ONLY \$249**



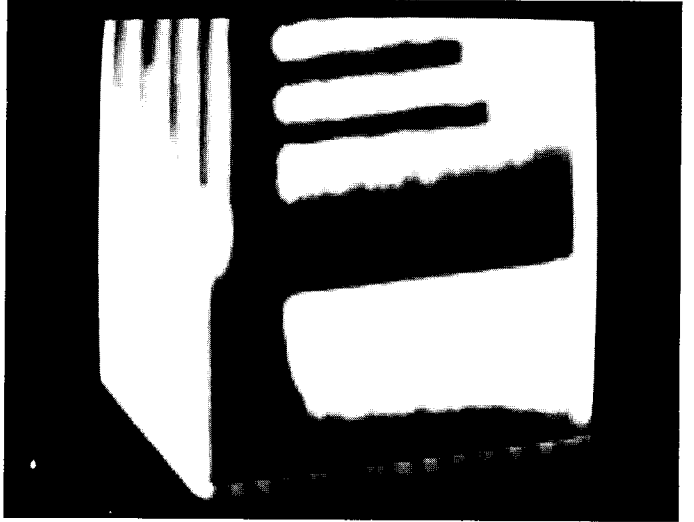
(a)



(b)



(c)



(d)

Figure 5—(a) shows the original 512x512 8-bit (256-gray-level) test image that is compressed, transmitted, and then reconstructed. The reconstruction is shown in (c). Both (a) and (c) were zoomed by a factor of four so a quick comparison of 'visual detail' between the original (b) and the reconstruction (d) can easily be made. Note the slight blurring, which is expected, because I pushed the quantization utility to its limit. Of course, the nature of the original image helped; it started out with 7.1% of its pixels set to a gray level of zero. This represented null DCT blocks for just over 40% of the image. The remaining 8x8 blocks were DCT quantized and then the whole image was encoded. I achieved a compression ratio (for this specific example) of 2.134:1.

monochrome 512 x 512 graphics image transfer). The only real difficulty that I encountered was in modifying the **Huffman** encoding procedure to improve reduction of redundancy in coefficients, especially zeros. So, when you get to this point, **have** fun. I found several schemes suggested, but in the end I just winged it, but with some success. My reconstructions are for the most part fairly good, but there is a little blurring of sharp distinctive edges if I set my visualization frequency band too narrow (see Figure 5). I usually found I eliminated too many of the high-frequency components in the quantization process.

**Well...have** at it. Writing your own code isn't too difficult or complicated, just be sure to use some sophisticated file access techniques to minimize image data access from your **HDD**. ❖

*Chris Ciarcia has a Ph.D. in experimental nuclear physics and is currently working as a staff physicist at a national lab. He has extensive experience in computer modeling of experimental systems, image processing, and artificial intelligence.*

## IRS

- 201 Very Useful
- 202 Moderately Useful
- 203 Not Useful

## REFERENCES

1. Livny, I. and D. Seltz, 'Vector Signal Processor-Based Image Processing Applications,' Advanced Printing of Paper Summaries, Electronic Imaging '89, Boston, Mass., Oct 2-5, p 179-184, 1989
2. Lhuillier, J.J and A.Q. Nguyen, 'Sub-band Coding of Images-Comparison with DCT,' Signal Processing, 4, 1988.
3. Huffman, A., 'A Method for the Construction of Minimum Redundancy Codes,' Proc. IRE, 40 1098-1101 (1952).
4. Dubois, E. and J.L. Moncet, 'Encoding and Progressive Transmission of Still Pictures,' IEEE Trans. Commun., vol COM34, pp 310-319.
5. Dubois, E., Y. Rahmouni, and F. Lortie, 'Experiments on Image Coding with Distortion Below the Visual Threshold,' SPIE, vol 845, pp 216-231.

Alfred L. Schumer

# Extended Serial Communications on the 8096

Increase the Utility of these Ubiquitous Chips with Simple C Software

When Intel designed the 8051 embedded controller, and later its 16-bit cousin the 8096, their serial input/output (SIO) capabilities were primarily intended for host and multiprocessor communications. Today, however, embedded microprocessors are being used in ways not originally envisioned, including SIO input filtering, often to peripheral devices with fixed and unusual framing characteristics. (In asynchronous serial communications, framing denotes the number of start, stop, data, and parity bits.)

This article examines the serial communications capabilities of the 8096 (and by default the functionally identical 8051) and develops additional SIO framing capabilities through a collection of C language functions. In addition, I'll discuss SIO deficiencies in Intel's implementation of the 8096 standard library and how these functions can be incorporated into embedded C applications to provide robust serial communications.

## HARDWARE SIO MODES

The 8096 provides four SIO modes in hardware including one synchronous and three full-duplex asynchronous modes, one of which is dedicated to multiprocessor communications. Mode 0 is a synchronous mode commonly used for shift-register-based I/O expansion in which eight bits are shifted out to the TXD pin, least-significant bit (LSB) first. Mode 1 is the standard communications mode consisting of ten bits: a start bit (0), eight data bits (LSB first), and a stop bit (1). If parity is enabled, an

even parity bit is sent instead of the eighth data bit and checked on reception. Mode 2 is the asynchronous ninth-bit recognition mode commonly used for multiprocessor communication. It consists of a start bit (0), nine data bits (LSB first), and a stop bit (1). The ninth data bit is programmable and is cleared after each transmission. During reception, the serial port receive interrupt will not be set unless the ninth bit is set. Finally, mode 3 is the asynchronous ninth-bit mode which is identical to mode 2 except that the ninth bit can be programmed as even parity and will always generate a receive interrupt on reception.

The serial port is controlled through the Serial Port Control register (SP\_CON) and the Serial Port Status register (SP\_STAT) both located at 11 h in the Special Function Register (SFR) file. The format for this SFR is given in Figure 1.

Writing to location 11h accesses SP\_CON, while reading it accesses SP\_STAT. Reads of SP\_CON return indeterminate data on the lower five bits, while writing to SP\_STAT has no effect. The TB8 bit of SP\_CON is cleared after each transmission and both TI and RI are cleared whenever SP\_STAT is read.

Excluding modes 0 and 2 due to their dedicated framing formats, general-purpose SIO is relegated to modes 1 and 3, which limit framing formats to one stop bit and even parity. Ideally, a robust SIO capability should provide one or two stop bits, seven or eight data bits, and odd, even, or no parity.

## SOFTWARE SIO MODES

Within certain constraints, soft-

SP_CON Bits	Description (Write On/y)
0	Bit 1 and bit 0 specify the mode: 00 = mode 0 10 = mode 2 01 = mode 1 11 = mode 3
1	
2	PEN enables the parity function
3	REN enables the receive function
4	TB8 programs the ninth data bit
SP_STAT Bits	Description (Read Only)
5	TI is the transmit interrupt flag
6	RI is the receive interrupt flag
7	RB8 is the ninth data bit received (if no parity) or: RPE is the parity error indicator (if parity)

Figure 1—The 8096's serial port is controlled through the Serial Port Control register (SP\_CON) and the Serial Port Status register (SP\_STAT).

ware can largely overcome the SIO hardware deficiencies of the 8096. Figure 2 lists the nine possible framing formats combining up to two stop bits, seven or eight data bits, and full parity generation (not all formats can be implemented due to an overall limit of 11 framing bits). These nine formats are identified as software modes zero through eight in the code example later on.

Implementing the nine software SIO modes is a straightforward exercise in bitwise manipulation utilizing C's bitwise operators AND (&), OR (|), and complement (~). Special care, however, must be given to working around

the write-only and read-only attributes of the Special Function Registers.

#### S1096.H AND S1096.C

Listings 1 and 2 contain the code to implement the software SIO modes. Compiled fully optimized with Intel's iC96, the ROMable code size is about 350 bytes and uses three dedicated registers.

SIO96.H is the header file to be included in applications using the software SIO modes. It defines constant identifiers for the nine modes (SP\_XMODE0-8) and function prototypes for five functions constituting low-level serial port control.

SIO96.C begins by referencing the external SFR registers sbuf, sp\_con and sp\_stat. It then defines local variables sp\_con\_save and sp\_stat\_save, which are shadow registers required to compensate for the read/write-only attributes of their counterparts. The variable sp\_mode holds the software SIO mode after initialization and is referenced during serial port reads and writes to ensure correct operation.

The function EvenParity and its related data structures parity\_table and parity\_masks comprise the methodology used to generate parity for seven- or eight-bit bytes. The method chosen favors execution speed over code size by using a look-up table containing even parity bits. The table is accessed within EvenParity by shifting the data value four bits to the right to locate the appropriate word which is then bit-masked based on the lower four bits. Boolean

Software Mode	Data Bits	stop Bits	Parity	Hardware Mode	PEN	Comments
0	7	1	even	1	1	hardware mode 1
1	7	1	odd	1	0	bit 7 = odd
2	7	2	none	1	0	bit 7 = 1
3	7	2	even	3	0	TB8 = 1, bit 7 = even
4	7	2	odd	3	0	TB8 = 1, bit 7 = odd
5	8	1	none	1	0	hardware mode 1
6	8	1	even	3	1	hardware mode 3
7	8	1	odd	3	0	TB8 = odd
8	8	2	none	3	0	TB8 = 1

Figure 2-The 8096 can be programmed to support nine possible framing formats, each identified as a software mode numbered zero through eight as described in the article.

true values returned indicate even parity.

SioInit initializes the serial port based on the software SIO mode passed in mode and enables the receive function if rcv is true. It must be called prior to using the other four functions and can be called any time thereafter to change software SIO modes. Since many of the software SIO modes share common hardware SIO modes, they are grouped together to save code space. The translation from software to hardware SIO mode corresponds to Figure 2 which should be referred to if the code is unclear. Finally, SioInit initializes the shadow registers and enables the TXD pin through the SFR IOC1 before returning.

Serial port reads and writes are accomplished with the low-level functions SioRead and SioWrite.

SioWrite conditions the serial port register SP\_CON and inserts appropriate parity bits in either the eighth or ninth bits as required. Equally important, it updates the shadow

registers sp\_stat\_save indicating the port is ready for writing again before actually writing out the data.

sioRead is considerably simpler than SioWrite, though its complexity would be increased substantially if parity checking were incorporated. After updating the shadow register sp\_stat\_save to indicate reception, it simply returns the value read from the port with the most-significant bit masked off for seven-bit data formats.

SioXmitRdy and SioRecvRdy return the status of the transmit and receive interrupt flags, respectively. It is important to note that each function first ORs the shadow register sp\_stat\_save with sp\_stat to preserve the RI and TI flag bits between reads. SioXmitRdy and SioRecvRdy should be called prior to SioWrite and SioRead to avoid overrunning the serial port.

```

/*****
*   sio96.h
*   Intel 8096 Extended Serial I/O Mode
*****/

#define sfr extern volatile register /* special register file */

#define SP_XMODE0 0 /* 7 data, 1 stops, even */
#define SP_XMODE1 1 /* 7 data, 1 stops, odd */
#define SP_XMODE2 2 /* 7 data, 2 stops, none */
#define SP_XMODE3 3 /* 7 data, 2 stops, even */
#define SP_XMODE4 4 /* 7 data, 2 stops, odd */
#define SP_XMODE5 5 /* 8 data, 1 stops, none */
#define SP_XMODE6 6 /* 8 data, 1 stops, even */
#define SP_XMODE7 7 /* 8 data, 1 stops, odd */
#define SP_XMODE8 8 /* 8 data, 2 stops, none */

typedef unsigned char byte; /* 8-bit register/scaler */
typedef unsigned shortword; /* 16-bit register/scaler */

void SioInit(byte, byte); /* init extended eio port */
void SioWrite(byte); /* extended sio mode write */
byte SioRead(void); /* extended sio mode read */
byte SioXmitRdy(void); /* boolean xmitter ready? */
byte SioRecvRdy(void); /* boolean receives ready? */

```

listing 1—SIO96.H is the header file included in applications using the software SIO modes.



```

/*****
*   aio96.c
*   Intel 8096 Extended Serial I/O Modes
*****/
#include "sio96.h96"

#define SP_MODE0 0x00 /* 8096 native sio mode 0 */
#define SP_MODE1 0x01 /* 8096 native sio mode 1 */
#define SP_MODE2 0x02 /* 8096 native sio mode 2 */
#define SP_MODE3 0x03 /* 8096 native sio mode 3 */
#define SP_TB8 0x10 /* ninth bit in sio mode 3 */
#define SP_PEN 0x04 /* parity enable modes 1/3 */
#define SP_REN 0x08 /* receive enable mode 1/3 */
#define SP_XMIT 0x20 /* transmit interrupt flag */
#define SP_REC V 0x40 /* receiver interrupt flag */

sfr byte sbuf; /* register 7: read/write */
sfr byte sp_con; /* register 17: write only */
sfr byte sp_stat; /* register 17: read only */
sfr byte iocl; /* register 22: write only */

static const word paritytable[16] = {0x9669, 0x6996, 0x6996,
0x9669, 0x6996, 0x9669, 0x6996, 0x6996, 0x9669,
0x9669, 0x6996, 0x9669, 0x6996, 0x6996, 0x9669};
/* bit set if even parity */

static const word paritymasks[16] = {0x0001, 0x0002, 0x0004,
0x0008, 0x0010, 0x0020, 0x0040, 0x0080, 0x0100, 0x0200,
0x0400, 0x0800, 0x1000, 0x2000, 0x4000, 0x8000};
/* masks for bits 0 - 15 */

static byte sp_con_save; /* shadow write only reg */
static byte sp_stat_save; /* shadow read only reg */
static byte sp_mode; /* extended sio mode 0 - 8 */

static byte EvenParity(value)
byte value;
{
    return(paritytable[value >> 4] & paritymasks[value & 0x0f]);
}

void SioInit(mode, rcv)
byte mode, rev;
{
    switch(mode)
    {
        case SP_XMODE0: sp_con_save = SP_MODE1 | SP_PEN;
                        break;
        case SP_XMODE1:
        case SP_XMODE2:
        case SP_XMODE5: sp_con_save = SP_MODE1;
                        break;
        case SP_XMODE3:
        case SP_XMODE4: /* TB8 permanently set */
        case SP_XMODE8: sp_con_save = SP_MODE3 | SP_TB8;
                        break;
        case SP_XMODE6: sp_con_save = SP_MODE3 | SP_PEN;
                        break;
        case SP_XMODE7: sp_con_save = SP_MODE3;
                        break;
    }
    if (rev)
        sp_con_save |= SP_REN; /* enable receive function */
    sp_con = sp_con_save; /* save for future writes */
    sp_stat_save = SP_XMIT; /* initialize for writes */
    sp_mode = mode; /* save for write routine */
    iocl = 0x20; /* enable TXD pin for sio */
}

void SioWrite(data)
byte data;
{
    switch(sp_mode)
    {
        case SP_XMODE4: /* set TB8 & fall through */
                        sp_con = sp_con_save;
        case SP_XMODE1: /* fall through to set to odd */
                        if (!EvenParity(data))
                            break;
        case SP_XMODE2: data |= 0x80; /* force bit 7 high for odd */
                        break; /* parity/second stop bit */
    }
}

```

(continued)

Figure 2-20. *SIO96.C* contains all the code necessary to implement software SIO modes on the 8096.

The C96 Library supplied with the Intel C compiler for the 8096 contains many of the standard C I/O functions such as `printf`, `putchar`, and `getchar` which are written to use the serial port of the microcontroller on a polled basis. A problem arises when the `putchar` routine is called the first time. Since nothing has been transmitted, the `TI` bit cannot be set, thereby putting the routine in an infinite loop polling `SP_STAT` until `TI` is set. Another problem arises when `SP_STAT` is read since both the `TI` and `RI` flags are cleared each time. Since `getchar` also polls `SP_STAT` to see when a character has been received, it causes `TI` to be inadvertently cleared while checking for the `RI` bit to be set. When `putchar` is called again, this can cause the program to enter the infinite loop described above.

`SIO96.C` can be used to correct these bugs by rewriting the standard functions `putchar` and `getchar` as shown in Figure 3a and 3b.

These functions should be linked before the standard library `c96.LIB` in your application. Alternatively, you can replace the routines `putchar` and `getchar` in `C96.LIB` by compiling each into a separate object file along with `SIO96.c` and executing the commands in Figure 3c (be sure to make a backup copy of `C96.LIB` before making any changes).

## NEW MODES FOR NEW APPLICATIONS

The 8096 (and the functionally equivalent 8051) support hardware serial input/output modes primarily intended for host and multiprocessor communications. Through the use of demonstrated software, these modes can be extended to provide multiple framing formats common among computer peripherals. In addition, the routines discussed fix a potentially fatal bug in the Intel standard C library for the 8096. The software is functionally compact and when used with embedded C applications on the 8096 provides a robust SIO communications capability.

## 8031 $\mu$ Controller Modules

NEW!!!

### Control-R II

- ✓ Industry Standard 8-bit 8031 CPU
- ✓ 128 bytes RAM / 8 K of EPROM
- ✓ Socket for 8 Kbytes of Static RAM
- ✓ 11.0592 MHz Operation
- ✓ 14/16 bits of parallel I/O plus access to address, data and control signals on standard headers.
- ✓ MAX232 Serial I/O (optional)
- ✓ +5 volt single supply operation
- ✓ Compact 3.50" x 4.5" size
- ✓ Assembled & Tested, not a kit

\$64.95 each

### Control-R I

- ✓ Industry Standard 8-bit 8031 CPU
- ✓ 128 bytes RAM / 8K EPROM
- ✓ 11.0592 MHz Operation
- ✓ 14/16 bits of parallel I/O
- ✓ MAX232 Serial I/O (optional)
- ✓ +5 volt single supply operation
- ✓ Compact 2.75" x 4.00" size
- ✓ Assembled & Tested, not a kit

\$39.95 each

#### Options:

- MAX232 I.C. (\$6.95ea.)
- 6264 8K SRAM (\$10.00ea.)

#### Development Software:

- PseudoSam 51 Software (\$50.00) Level II MSDOS cross-assembler. Assemble 8031 code with a PC.
- PseudoMax 51 Software (\$100.00) MSDOS cross-simulator. Test and debug 8031 code on your PC!

#### Ordering Information:

Check or Money Orders accepted. All orders add \$3.00 S&H in Continental US or \$6.00 for Alaska, Hawaii and Canada. Illinois residents must add 6.25% tax.

#### Cottage Resources Corporation

Suite 3-672, 1405 Stevenson Drive  
Springfield, Illinois 62703  
(217) 529-7679

Reader Service #120

```

case SP_XMODE3: sp_con = sp_con_save; /* set TB8 high */
                 if (!EvenParity(data)) /* bit7 even par */
                     data |= 0x80;
                 break;
case SP_XMODE7: if (EvenParity(data)) /* TB8 is odd par */
                 sp_con_save |= SP_TB8;
                 else
                     sp_con_save &= ~SP_TB8; /* set TB8 */
case SP_XMODE8: sp_con = sp_con_save; /* TB8 on in xmode8 */
                 break;
}
sp_stat_save &= ~SP_XMIT; /* clear xmit ready flag */
sbiif = data; /* write data to sio port */
}

byte SioRead()
{
    sp_stat_save &= ~SP_RECV; /* clear rcv ready flag */
    if (sp_mode > SP_XMODE4) /* 8 data bit modes only */
        return(sbuf);
    else
        return(sbuf & 0x7F); /* mask parity/stop bit 7 */
}

byte SioXmitRdy()
{
    /* sp_stat cleared on read */
    return((sp_stat_save &= SP_XMIT) & SP_XMIT);
}

byte SioRcvRdy()
{
    /* sp_stat cleared on read */
    return((sp_stat_save &= SP_RECV) & SP_RECV);
}

```

listing 2-continued

<p>a) <pre> int putchar(c) int c; {     while (!SioXmitRdy())         ;     SioWrite(c);     return(c); } </pre></p>	<p>b) <pre> int getchar() {     while (!SioRcvRdy())         ;     return(SioRead()); } </pre></p>
<p>c) LIB96 ADD SIO96.OBJ TO C96.LIB</p> <p>LIB96 REPLACE PUTCHAR.OBJ,GETCHAR.OBJ IN C96.LIB</p>	

**Figure 3**—Bugs found in the Intel C96 library may be fixed by rewriting the `putchar` (a) and `getchar` (b) routines. The new routines may be included in the library using two librarian commands (c).

#### Further Reading

Intel Corporation. **The 16-Bit Embedded Controller Handbook**. Santa Clara, CA: Intel Corporation, 1989.

Intel Corporation. **iC96 Compiler User's Guide**. Santa Clara, CA: Intel Corporation, 1988.

Katz, Ron, and Boyet, Howard. **The 16-Bit 8096: Programming, Interfacing, Applications**. New York, NY: Microprocessor Training, Inc., 1986.

Peatman, John B. **Design with Microcontrollers**. New York, NY: McGraw-Hill Book Company, 1988.

*Alfred Schumer* was formerly Chief Financial Officer of *Summagraphics Corporation*. An avid programmer and recent hardware neophyte, he recently relocated to Seattle for a change of lifestyle and to pursue other interests.

#### IRS

204 Very Useful  
205 Moderately Useful  
206 Not Useful

## Part 1

John Dybowski

# ONDI-The ON-line Device Interface

*Building a Powerful Remote Control for Your PC*

**C**ontrolling computers remotely via modem is coming into vogue. There are a number of software packages available that allow a user at a remote site to log on, access the command line, and exchange files with a host computer. All require the host to be powered up and running special software in order to be accessible. These requirements can cause serious problems when remote control is required, so I started looking for more reasonable, full-featured solutions.

As is often the case, a better balance of features and performance can be obtained by combining hardware and software; what we need is an *ON-line Device Interface: ONDI*. ONDI is a hardware device that can manage the key computer **interface elements**—the keyboard, COM port, and computer power—under remote control. Due to the popularity of IBM PC-type computers, this project will focus on an interface device specifically designed to work with IBM (or compatible) PC/XT and AT computers (or PS/2s with a cable adapter). ONDI, along with a Hayes-compatible modem, will allow a remote computer to log on to the device, control power to the computer, simulate keyboard input, and allow direct communications to the computer. Password protection, line monitoring (for idle terminals and special escape sequences), and various other commands that can be invoked locally or remotely will round out the feature set that will be discussed in detail in the next issue when we cover the software side of ONDI.

## DECISIONS, DECISIONS...

First we must make some decisions on implementing the hardware. Electrically, this project will involve a microcontroller, keyboard interface, serial interface, power relay, miscellaneous indicators and switches, and a power supply. To implement **the logic** for the system, we need a **microcontroller** capable of modest performance, a UART, some parallel I/O, program memory, battery-backed data memory, and a program watchdog. A reasonable choice would be an Intel 8031. Now, with the 8031 it's always the same old story: everyone seems to agree that the part is a pain to use, but it's hard to resist the savings in parts count that the on-chip peripherals provide. The bottom line is, there's a lot of grumbling and a lot of 8031s being sold. The main problem with the 8031 is that, in spite of the integration of peripheral functions, many designers use the part in its external memory mode that requires more parts and uses more I/O pins which, in turn, generally forces the use of more parts to restore the lost I/O.

## THE GUYS IN THE BACK ROOM

Now everyone can have a personal research and development team. **.you** know, the guys they keep locked away in some back room. Two outfits in particular come to mind that have consistently produced what could be considered widgets and inventions. Dallas Semiconductor provides the answer to the microcontroller problem in the form of the **DS2250** by

integrating all the requirements mentioned above on a 0.84" by 2.65" prefab "**SipStik**." The DS2250 is instruction set compatible with the Intel 8051 but integrates on-chip many of the features needed in most control applications: program and data memory, lithium backup power for memory, watchdog timer, and perhaps most important, an embedded serial program loader.

One of the most notable features of the DS2250 is its close resemblance to the 8051, quirks and all. Perhaps it would have been a good idea to enhance the instruction set or to provide an extra data pointer, but Dallas exhibited tremendous restraint in rendering the part. This could be a valuable lesson: it's all too easy to get buried in myriad features of dubious usefulness. The density the DS2250 affords sets the tone for a compact circuit design which is further enhanced with the aid of a few parts from Maxim. The MAX612 integrated AC-to-DC power converter and MAX233 single-chip dual 5-volt-only RS-232 transceiver. We'll throw in some mundane so we don't have too much fun and tie everything together with some HC logic. Now let's look at the hardware.

## THE HARDWARE

The controller is an **8-MHz** DS2250 with **8K** of embedded RAM. The **8-MHz** version is the least expensive and provides adequate performance for this project. The crystal frequency of 5.5296 MHz is used because the DS2250 automatically determines the

baud rate to use for communications when in program load mode. Since 5.5296 MHz is half the standard frequency of 11.0592 MHz, the controller can operate at standard baud rates using the program load facility at the host computer. The power-on reset is implemented within the DS2250, so no external components are required for this function.

Linepower at 110VAC is brought directly onto the main board and is stepped down (and isolated) to 8 VRMS, rectified, and regulated to +5 VDC. A Maxim MAX612 power converter provides rectification and regulation, keeping the power supply component count low. The '612 contains a full-wave bridge rectifier, an 18-volt zener diode, and a 5-VDC series regulator in an B-pin DIP package.

The addition of a Signal PC16-558-VRMS transformer, a 470- $\mu$ F filter capacitor, and a peak-input current-limiting resistor provides a 100-mA, +5-V,  $\pm 4\%$  power supply that will operate at a line voltage of from 80 to 160 VRMS. There is no power switch-the intent is to have power present to the device at all times.

Computer power is controlled via an optically isolated solid-state relay (SSR). The power SSR is a IO-amp 120-VAC unit with built-in snubber. A 1k series resistor is connected between 5 V and the positive control terminal in order to limit control current to the minimum level required for reliable operation. Since the required control current is kept low, the return control terminal is driven directly from PO.4 of the DS2250.

The mode switch is a three-position, single-pole/double-throw toggle-type with center off. The common is tied to ground with the switched contacts connected to PO.6 and P3.5. Note that the switch body is tied to earth ground to divert any static discharges from the unbuffered switch connections to the DS2250.

Four LEDs are used to indicate status information. These are HLMP 4700 low-current types that operate at 2 mA. Three of the LEDs are under software control and are driven directly by PO.0, PO.1, and PO.2 of the DS2250. The fourth LED denotes program load mode and is connected directly to the program load switch.

The program load switch is a triple-pole/single-throw switch used to place the DS2250 into the program

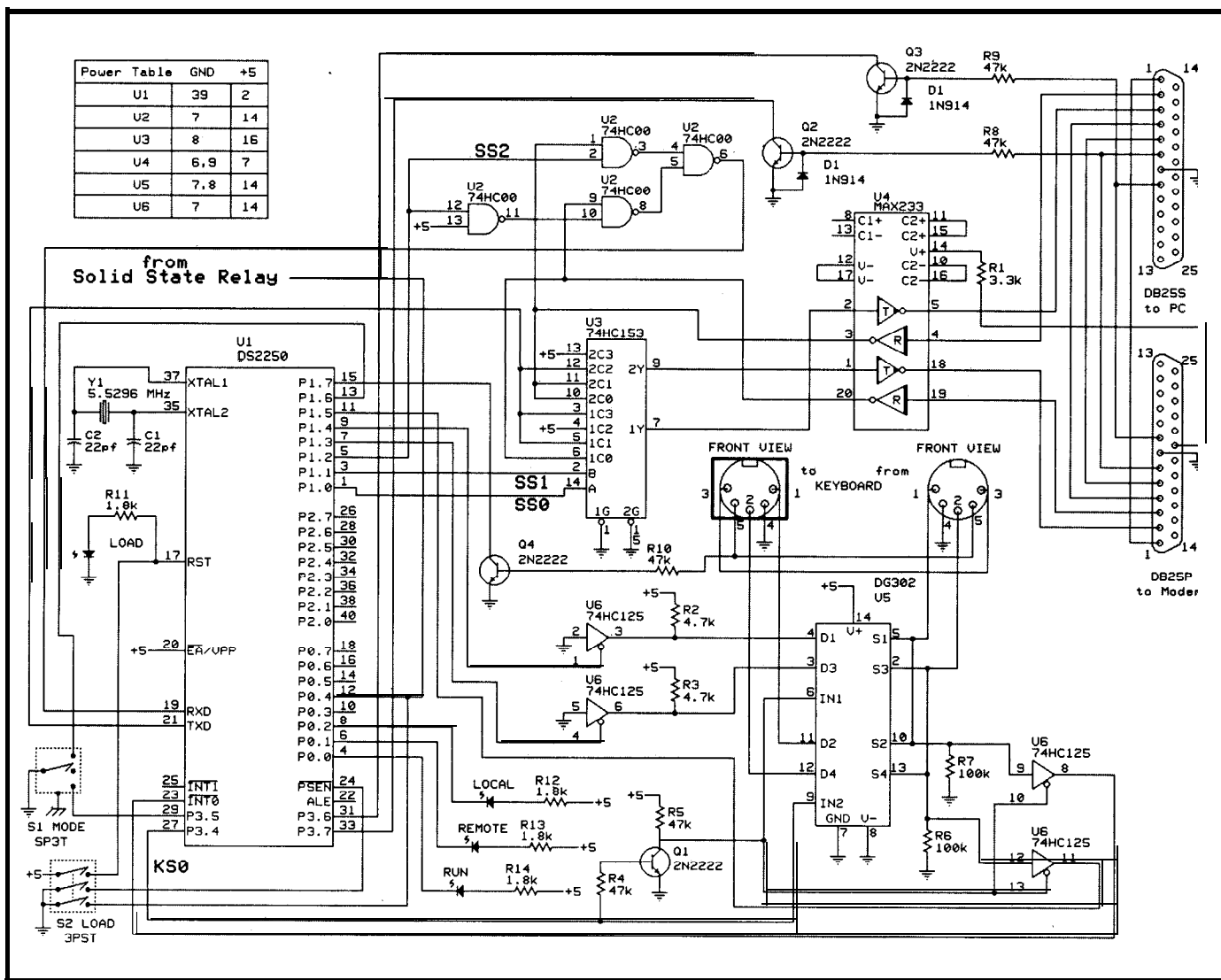


Figure 1—The complete ONDI schematic shows connections for two serial ports, keyboard input and output ports, and a number of configuration switches and lights.

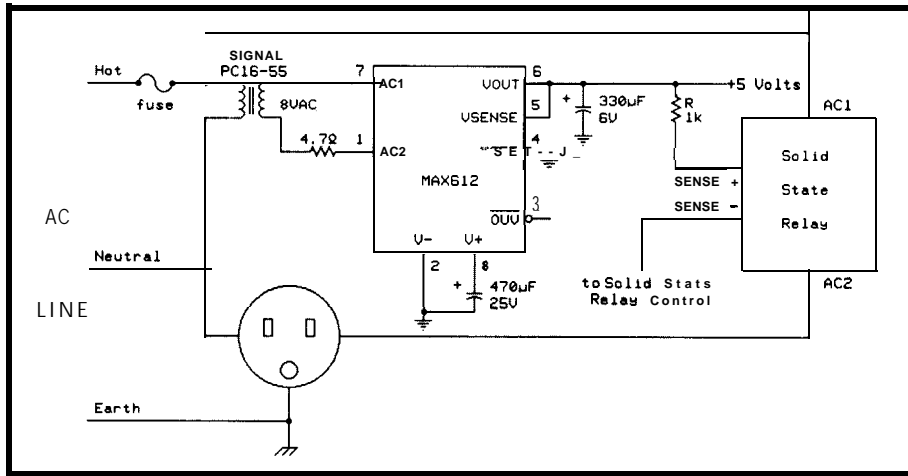


Figure 2-A MAX612 is used to regulate 5 volts from the power line, while the computer's power is switched using a solid-state relay.

load mode and energize the power SSR. The three poles of the switch are connected as follows: 5 volts is brought out to the DS2250 RST pin and to the anode of the load LED. Ground is brought to PSEN\, and an independent ground connection is brought out to the return control terminal of the power SSR. The DS2250 enters program load mode when PSEN\ is brought low and RST is simultaneously brought high. This mode is indicated by the load LED being lit. It is assumed that program loading will be performed from the host computer, therefore the power SSR is energized when the switch is placed into the load mode, overriding the power SSR control pin from the DS2250.

#### KEYBOARD INTERFACE

The IBM keyboard electrical interface consists of a clock and a data

signal. The procedure used to transmit a bit is to set the data line to the appropriate level, then assert the clock low, then bring it high again. The data is valid from before the falling edge to after the rising edge of the clock. The bit transmission is the same for both the XT- and AT-style keyboards. The number of bits per scan code, method of indicating make/break codes, idle signal states, and protocol differ depending on the keyboard style.

Data from the keyboard normally passes transparently through to the computer, but the DS2250 can also seize control of the keyboard port, switch out the keyboard, and supply its own simulated keyboard data stream to the computer. Switching is accomplished with a DG302 dual double-pole/single-throw solid-state analog switch. Each switch has an associated control line. The seize signal, KSO, is driven from P3.4 and is

connected to the control line of one of the DG302's control lines, and to the other control line through a 2N2222 transistor, which functions as an inverter. Wired in this way, the DG302 functions as a dual double-pole/double-throw switch. The analog switch is required since the keyboard communication protocol is defined as bidirectional; the switch allows signals to pass in either direction. The DS2250 drives the keyboard clock and data signals with two gates of a 74HC125 tristate buffer arranged to function as an open-drain device.

The keyboard clock and data at the computer interface are received via the other two gates of the 74HC125 and are presented to INTO\ and P1.5 respectively. The receive section is set up to disable reception of clock and data while the DS2250 is transmitting its simulated keyboard data stream; this is intended to simplify the interrupt-driven receive routine. The clock and data lines are tied to ground at the computer interface via 100k resistors in case the device is operated without its keyboard interface connected (to prevent float conditions). The computer's 5-V logic supply is present on the keyboard interface and is monitored by the DS2250 via a common-emitter transistor arrangement on P0.7 to determine when the computer is actually powered up.

#### SERIAL INTERFACE

The serial interface is the heart of the system and contains the drivers

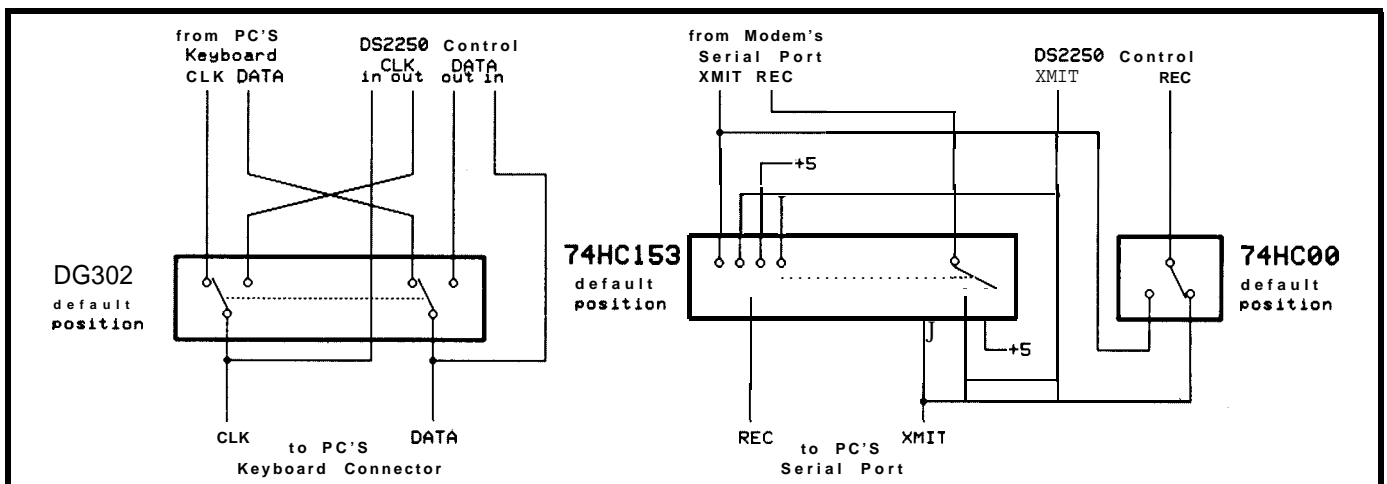


Figure J-Redrawing the multiplexers in the form of switches helps clarify their operation.

## THE DS2250

The **DS2250** is a CMOS **8-bit** microcontroller that emulates the functions of an 8051. The program/data memory space is implemented using nonvolatile CMOS static **RAM** that resides on its own internal embedded memory bus. This leaves all on-chip parallel I/O available for use as I/O. Nonvolatile operation is accomplished via an embedded lithium power source that will maintain the nonvolatile areas of the **DS2250** for 10 years in the absence of primary power. Initial loading of the application software into the **DS2250** is possible from either a parallel or serial interface to the host system. This function allows initialization of the nonvolatile areas of the device including program/data RAM and the configuration parameters. The **DS2250** incorporates control functions which provide crashproof operation when system power is momentarily disrupted or removed. These functions include the power fail warning interrupt, automatic power down, and power on restart. Regardless of the duration of the power outage, the **DS2250** has the ability to resume execution when power is restored as if the power failure had not occurred at all.

### REGISTERS

All CPU registers are mapped as special function registers (**SFRs**) and are identical in number and function to those present in the 8051. The power control (**PCON**) register is in the same location as in the 8051. All of the bits in this register remain unchanged in the **DS2250** implementation with the following exceptions: the **GF1** (**PCON.3**) and **GFO** (**PCON.2**) flags have been changed to enable the power fail interrupt and enable the watchdog timer reset mask bits, respectively. The previously unused **PCON.5** bit has been assigned as the power fail (**PF**) flag which may be polled by the application program to test for a power fail condition. The **MCON** register is used to define the total range of memory in the embedded RAM and the respective amounts of program and data memory within the embedded RAM. The timed access (**TA**) register interfaces to logic that is used to prevent access to key internal resources in the event of errant program execution. These resources include the partition address in the **MCON** register and the watchdog timer control bits in the **PCON** register.

### EMBEDDED PROGRAM/DATA RAM

Current versions of the **DS2250** incorporate from **8K** to **64K** of RAM on the embedded memory bus. On-chip logic allows this memory to be partitioned during initial loading of application software as program memory or data memory. This must be done in order for the **DS2250** to begin execution of the application software from the space in which it has been loaded. The **DS2250** preserves the Harvard memory architecture of the 8051, in which there are separate areas for program and data memory. The **DS2250** does not permit the overlapping or combining of program and data memory; the partition address is the point at which program memory ends and data memory begins. At first this may seem like a disadvantage since it would appear that unnecessary gyrations must be incurred in accessing the separate regions. Actually this is an important safety feature. Since the executable program resides in RAM, errant program execution could corrupt the program itself if it were possible to write to this region. The **DS2250** enforces the separation of address spaces; therefore, since no 8051 instruction exists for writing to program memory, it is impossible for the **DS2250** to modify its program while executing (unless the partition address becomes reset).

### PARALLEL I/O

Four **SFRs** provide access for the four parallel I/O port latches (**PO, P1, P2, P3**). A total of 32 bits of parallel I/O is available through these I/O ports. All are available when embedded RAM is used for program and data memory.

### TIMED ACCESS LOGIC

Timed access logic is used to protect against inadvertent changes to program RAM and the configuration parameters in the event of a loss of software control. The protected configuration area includes the partition address bits in the **MCON** register, the watchdog timer enable bit, stop mode bit, and power-on-reset bit in the **PCON** register. These protected bits may only be written through the execution of two consecutive write operations within four machine cycles of each other to the timed access (**TA**) register. The first write operation must be with a value of **AAH** and the second with a value of **55H**. When this sequence is performed, write access is allowed to the protected bits, which must be written within four machine cycles of the double write sequence to the timed access register.

# INTROL CROSS DEVELOPMENT SYSTEMS

SAVE Development  
and Debugging Time  
of Embedded  
Microprocessor Systems!

- INTROL-C Cross-Compilers
- INTROL-Mcdule-2 Cross-Compilers
- INTROL Relocating Macro Cross-Assemblers

COMPILER PACKAGES INCLUDE:  
Compiler • Assembler • Linker  
• Runtime library, including a multi-tasking executive • Support utilities • Full year's maintenance

TARGETS SUPPORTED:  
6301/03 • 6801/03 • 6809 •  
68HC11 • 68000/08/10/12 •  
68020/030/881/851 • 32000/  
32/81/82

AVAILABLE FOR FOLLOWING  
HOSTS: VAX and MicroVAX;  
Apollo; SUN; Hewlett-Packard;  
Macintosh; Gould Power-  
Node; IBM-PC, XT, AT, and  
compatibles

INTROL CROSS-DEVELOPMENT  
SYSTEMS are proven, accepted  
and will save you time, money,  
and effort with your develop-  
ment. All INTROL products are  
backed by full,  
meaningful,  
technical support.  
CALL or WRITE  
for facts NOW!



**INTROL**  
CORPORATION

9220 W. Howard Avenue  
Milwaukee, WI 53228  
414/327-7171 FAX: 414/327-7734  
Quality Software Since 1979

and receivers as well as the **data-switching** circuitry. RS-232 data is received using a MAX233 line driver/receiver and all data manipulations are performed on the CMOS HC-level signals. Data received from either the modem or the computer is routed to the DS2250 via a multiplexer constructed from a 74HCO0. The routing is determined by the signal SS2 driven by P1.2. When this signal is at a logic 1, the DS2250 receives the computer transmission; when SS2 is a 0, the DS2250 receives the **modem transmission**. Outgoing data is presented to the 74HC153 dual 4-to-1 multiplexer. The computer-bound mux is **sourced** by the modem transmit, the DS2250 transmit, and a marking signal. The modem mux's inputs are from the computer transmit, DS2250 transmit, and marking signal. The SSO and SS1 signals, driven by P1.0 and P1.1 respectively, determine which signals are connected to the computer and modem receive lines. Table 1 contains a summary of the binary routing arrangement.

All the main RS-232 signal lines with the exception of DTR, receive, and transmit are passed directly from the modem port to the computer port. DTR is connected to the 10-V output of the MAX233 through a **3.3k** resistor and is always on. Also, the DS2250 has the capability of monitoring the more important RS-232 lines-DSR and particularly DCD. The DS2250 receives the signals using two **2N2222** common-emitter transistors and samples them on P3.6 (DCD) and P3.7 (DSR). The diode clamps on the base leads limit the negative swing from the RS-232 level signals.

## DEFAULT SIGNAL CONSIDERATIONS

In the design of embedded systems, the states that the system signals assume at power-up are an important consideration, particularly in the case of malfunctioning or nonfunctioning equipment. In this design, the **default states of the DS2250's pins are particularly important** because the DS2250 is essentially reset when it is in program load mode. The reset state criteria is defined as follows:

## WATCHDOG

**When user software is executing, the watchdog timer can be used to restart the processor in the event of errant program execution. When the watchdog timer is enabled, it will eventually timeout after a fixed number of clock cycles unless it is reset by the application software. An internal reset to the CPU is generated if the timeout condition is ever reached.**

## SERIAL PROGRAM LOADER

The DS2250 is placed in its program load configuration by simultaneously applying a logic 1 to the RST pin and forcing PSEN to a logic 0 level. Immediately following this the DS2250 will look for a serial ASCII carriage return (ODH) character received at 9600, 4800, 1200 or 300 bps over the serial port. A standard 11.0592-MHz timebase is required in order for communications to take place at standard baud rate frequencies. (A 5.5296-MHz crystal will also work since it is half the standard frequency and the DS2250 will be able to recognize the standard baud rates up to 4800 bps.) A software utility program for an IBM PC is available to configure and load the DS2250 without detailed knowledge of the operation of the DS2250's serial load command line syntax.

**Important safety tip: Leave P2.7 and M.6 open or pulled up during serial programming. Failure to do this results in parallel load operation.**

DS2250 transmit is routed to the computer. Computer transmit is routed to the DS2250. Keyboard is passed directly through to the computer. Front-panel LEDs are off. Power SSR is deenergized. When in program load mode, the load switch will turn on the load LED and energize the power SSR.

That's the hardware, with a little help from the guys in the back room... and ONDI. We now have a re-programmable interface device with some hooks that can be put to use in various situations. By virtue of the keyboard emulation and power control capabilities, we can devise applications where ONDI exercises control over the host computer. With some PC software, this presents the opportunity for distributing the processing burden. Although we can consider some interesting uses for the interface device, for the time being we'll stick with remote computer control. Next time we'll explore the system software for a remote control application, program downloading, and go into a

little detail on keyboard protocols. Unfortunately, things go downhill from here. When it comes to software, we can't call on the guys in the back room-we are the guys in the back room. ❖

*John Dybowski has been involved in the design and manufacture of hardware and software for industrial data collection and communications equipment. His crowning achievements are his daughter Ondi and his son John.*

## SOURCES

**Dallas Semiconductor Corp.**  
4350 Beltwood Pkwy South  
Dallas, TX 75244-3219  
(214) 450-0400

**Maxim Integrated Products, Inc.**  
120 San Gabriel Dr.  
Sunnyvale, CA 94086  
(408) 737-7600

## IRS

207 Very Useful  
208 Moderately Useful  
209 Not Useful

SS1	SS0	Computer Receive	Modem Receive
0	0	Modem Transmit	Computer Transmit
0	1	DS2250 Transmit	Computer Transmit
1	0	Marking Signal	DS2250 Transmit
1	1	DS2250 Transmit	Marking Signal

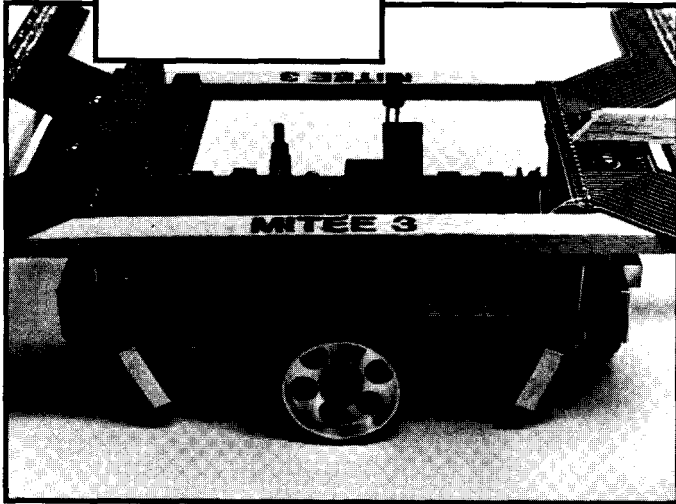
**Table 1** -The SSO and SS1 lines control the interconnections between the serial ports.

## Part 2

David Offen

# Building MITEE Mouse III

*The Software for a Maze-Running Rodent*



In Part 1 of this article I described the hardware for building a micro-mouse. Using MITEE Mouse III as an example, I described what type of systems are required and how they might be implemented. In this second article, I hope to breathe life into the hardware by describing the ideas behind the mouse software.

### MAZE SOLVER

One aspect of building a micro-mouse which attracts many people is the maze solver. This is not as difficult as is often perceived, however. The algorithm described here, which is variously known as *Bellman's algorithm* or the *flooding algorithm*, will find the optimal path through a maze once the configuration (walls and openings) is known. The initial problem, of course, is that the walls and openings are unknown. What is known, however, is that the maze is a 16 x 16 square array, MITEE Mouse III starts by building a map of the maze in its memory. Initially all the walls in the maze are marked as unknown. It then sets off toward the goal, assuming the unknown walls are open

(the most optimistic assumption). At every square that has any unknown walls surrounding it, the mouse stops and updates its maze map with the true information. When it sets off

toward the goal again, it always knows the true wall status in at least the square where it started, and is therefore able to advance at least one square.

Each time MITEE Mouse III sets off toward the goal, it runs the maze solver. Figure 1 shows a sample 4 x 4 maze with the start in square 00 and the finish in square 33. For the initial example, the shortest path is considered the optimal path. The objective of the algorithm is to fill each square of the maze with a number representing the minimum distance from that square, through the maze, to the goal. This is accomplished in a series of iterations. On each iteration, all the squares of a given distance are filled in. Since the next iteration will always fill in squares adjacent to the squares filled in on the last iteration, these

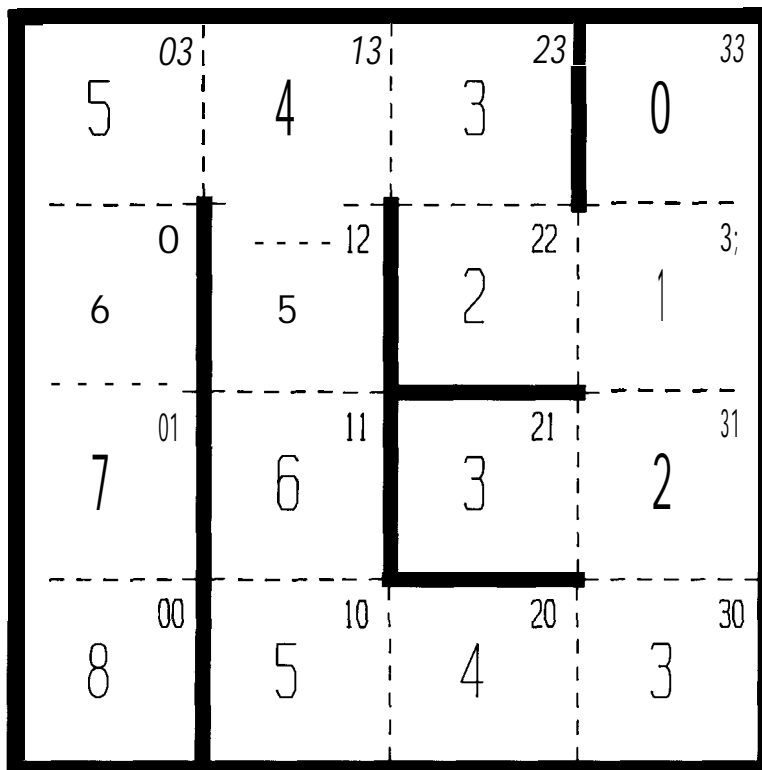


Figure 1 - A sample 4 x 4 maze.



squares are called tails and maintained in a list.

The solver starts by numbering the goal square with 0. On the first iteration, square 32 is given a number of 1 and put in the list of tails. It is the only square which is adjacent to the goal square and not separated from it by a wall.

**On the second iteration,**

square 32 is retrieved from the tail list and analyzed. Square 22 and square 31 are adjacent to square 32 and not separated by a wall, so they are given a number of 2 and replace the previous entry in the tail list. On the third iteration, square 23, square 21, and square 30 are given a number of 3 and put in the tail list. The fourth iteration deletes the tail at square 21 since there are no unnumbered squares adjacent to it that are not separated by walls. It also numbers square 13 and square 20 with a 4. On the sixth iteration, the two tails that meet at square 11 are combined into a single tail. This tail is then deleted during the seventh iteration because there are no unnumbered squares adjacent to it. The eighth iteration numbers square 00, the start square, which terminates the numbering process. The optimal path is now found by starting at the start square and proceeding to the square with the next lower number until the goal square is reached.

The maze solver always gives the best path based on the assumptions. When the optimal path does not take the mouse through any unknown walls, there are no more assumptions that need to be verified. The maze is now solved. There may be unknown walls left in the maze, and indeed finding the best path without fully mapping the maze is always a secondary goal, but they will not influence the path selected by the mouse.

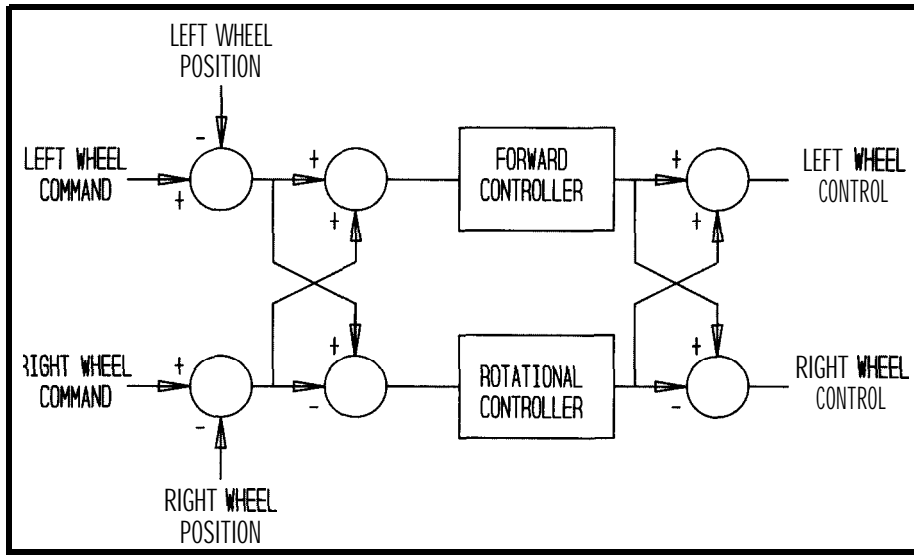


Figure 2—Separate controllers are used for forward and rotational movements, with the outputs combined to drive the motors.

This algorithm can be extended to compute the fastest path instead of the shortest path. This is done by filling the squares with the time to the goal instead of the distance to the goal. Many mice are able to accelerate on a long straight section so that the total time for  $N$  squares in a line is much less than  $N$  times the time for one square. This requires keeping track of the number of squares in a straight

section. To create two independent control signals, the motion is resolved into two modes: forward motion, where both wheels turn in the same direction; and rotational

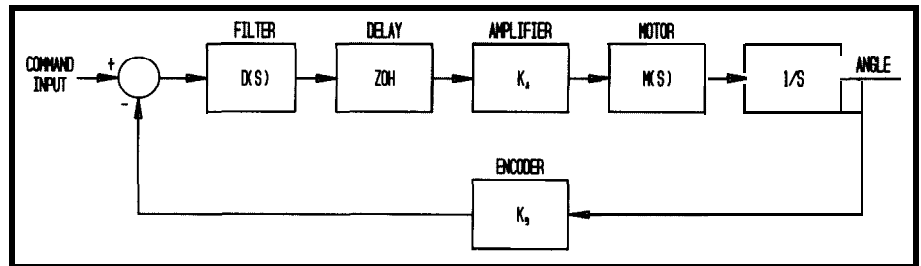


Figure 3—Each of the motor control loops can be represented for analysis purposes by a block diagram.

and knowing the incremental time for each additional square when filling in the time. In order to demonstrate the fastest-path capability of a mouse, most contest maze designs include straight sections that are several squares longer than the shortest path.

#### SERVO LOOP

The movement of MITEE Mouse III is controlled through two drive wheels. Differences in speed of the two wheels are used to steer the mouse, and changes in average speed of the two wheels are used to accelerate and

brake it. Good control of each wheel is clearly required to provide good overall control of the mouse in the maze.

MITEE Mouse III uses DC motors with incremental encoders to drive each wheel through a spur gear. The encoder measures the position of the motor shaft. Software in the mouse compares the actual position of the motor with the desired position, and computes an error. This error is used to drive the DC motor. Unfortunately the motions of the two wheels are not independent since they are connected together by the chassis of the mouse. To create two independent control signals, the motion is resolved into two modes: forward motion, where both wheels turn in the same direction; and rotational

# DEVELOPMENT TOOLS

For the IBM PC and Compatibles

## 2764-27256 ROM EMULATOR



Appears as EPROM to target system. Connects to PC parallel port.

- Plugs into target ROM socket and connects to PC parallel port via modular telephone cable
- Accepts 8K x 8 or 32K x 8 SRAM or NV SRAM
- Reset output restarts target after downloading
- Uses HC/HCT logic for CMOS & TTL compatibility
- Loads Intel Hex, Motorola S, hex, and binary files
- Command line software can be run from batch files for automatic downloading after assembly

\$ 99 (without memory)    \$119 (with 32K x 8 SRAM)    \$139 (with 32K x 8 NV SRAM)

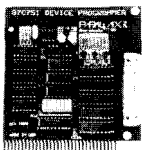
## 8051 FAMILY ASSEMBLER

- Works on all 8051 derivatives
- Supports standard Intel syntax
- Allows local labels and include files
- Labels may be up to 32 characters
- Generates assembly listings
- Outputs Intel Hex

**\$59**

Quick • Dependable • Clean Operation

## 87C751 DEVELOPMENT TOOLS



87C751 Programmer

Programs Signetics 87C751 and 87C752 microcontrollers. Handles EPROM, key, and security bits. Loads Intel Hex, ASCII hex, and binary files.

Programmer-DIP (includes DIP socket) 9299

Programmer - PLCC (includes PLCC socket) 9299



87C751 Bondout Board

"Mini-ICE" for the 87C751. Provides external EPROM socket for program memory. Very useful when combined with ROM emulator.

Bondout Board \$239

Bondout Board and ROM Emulator \$338

PARALLAX

(916) 721-8217

FAX: (916) 726-1905

Parallax, Inc.  
6200 Desimone Lane, #69A  
Citrus Heights, CA 95621

California residents add sales tax.  
Shipping: No charge for UPS ground,  
\$9.00 for UPS 2nd day, \$18.00 for UPS next day.



Reader Service #164

motor driving the load of the mouse are contained in  $M(s)$ . This term will be different for the forward and rotational controllers. The  $1/s$  term is due to the conversion from wheel speed to its integral, wheel position.  $K_d$  is the gain of the shaft encoder.

To analyze the  $M(s)$  term it is helpful to realize that the drive motors generate a torque, and this torque acts on an inertia. It is therefore convenient to convert the load on the mouse motors to an equivalent inertia.  $J_f$  is the inertia seen by the forward controller and is based on the mass of the mouse.  $J_r$  is the inertia seen by the rotational controller and is due to the reflected moment of inertia of the mouse.

$$J_f = \frac{M_m R_w^2}{2N^2} \quad J_r = \frac{J_m R_w^2}{2N^2 R_m^2}$$

where:

$M_m$  = mass of the mouse

$J_m$  = moment of inertia of the mouse

$R_w$  = radius of the drive wheel

$R_m$  = radius of the mouse (half the distance between the drive wheels)

$N$  = gear ratio from motor to drive wheel

The torque supplied by the motors is proportional to the motor current which in turn comes from the power amplifier through the motor resistance. This torque acts on the motor inertia, as well as the reflected inertia of the load, to create two time constants, one for each mode.

$$\tau_f = \frac{R(J_a + J_r)}{K_t^2} \quad \tau_r = \frac{R(J_a + J_f)}{K_t^2}$$

where:

$J_a$  = inertia of the motor armature

$K_t$  = torque constant of the motor

$R$  = resistance of the motor

In addition to the mechanical time constants, there is also an electrical time constant due to the inductance and resistance of the motor winding. This is combined with the back EMF of the motor to provide the overall transfer function.

$$M(s)_f = \frac{1/K_e}{(s\tau_f + 1)(s\tau_e + 1)}$$

$$M(s)_r = \frac{1/K_e}{(s\tau_r + 1)(s\tau_e + 1)}$$

where:

$K_e$  = back EMF of the motor

$\tau_e$  = electrical time constant of the motor

The previously mentioned lead-lag filter  $D(s)$  was added to the feedback loop to improve its bandwidth and stability. The design of the filter was done with a computer-aided design program called **MathCAD**. In addition to calculating the pole and zero placements of the filter, this program also converts the **continuous-time design to a discrete-time implementation**. The discrete-time implementation accounts for the fact that the control signal to the motor is only updated periodically. Each new control output to the motor is based on a new error measurement, the error measurement of the previous sample, and the control output of the previous sample.

$$\begin{aligned} \text{new output} &= K_1 \times \text{new error} \\ &\quad - K_2 \times \text{old error} \\ &\quad + K_3 \times \text{old output} \end{aligned}$$

The constants  $K_1$ ,  $K_2$ , and  $K_3$  are calculated by **MathCAD** from the continuous-time filter parameters and the sampling rate.

The 78312 processor used on MITEE Mouse III has a 16-bit multiply instruction but unfortunately it is unsigned. The **sign of the product** must be determined by software. Only the error terms are signed, however; the constants are all positive. This allows some savings in code size and execution time over a full signed multiply routine. MITEE Mouse III uses a 1-kHz sampling rate and requires approximately 86  $\mu$ s to execute the **motor control subroutines**. Though probably not necessary, in-line coding instead of subroutines is used to minimize the execution time.

The servo loop is very good at correcting errors in wheel position, but it can only do this after they occur. Certain errors however, are **predict-**

able and can be anticipated. To correct them, a feed-forward term is added into the voltage driving the motor. Feed-forward terms for static loss such as bearing and brush losses, as well as velocity and acceleration were implemented.

The gain of the power amplifier  $K_a$  depends on the battery voltage. During a run through the maze the battery voltage may vary from 12 to 9 volts. To minimize the effect this has on the gain of the feedback loop, as well as the feed-forward compensation, the battery voltage is continuously measured. This information is then used to hold the overall gain of the amplifier constant. [Editor's Note: **Software for this project is available on the Circuit Cellular BBS and on Software On Disk #16. See page 85 for downloading and ordering information.**]

#### PROFILE GENERATOR

After the servo loop is completed and the wheels can be moved to a commanded position, there remains the task of coming up with a series of positions which fulfills the other requirements of the mouse. The software which does this is called the

profile generator. There are two main types of maneuvers the mouse must perform: straight runs and turns. In general, the mouse will try to do both of these in the least amount of time. The major limitation to the speed of the mouse seems to be wheel traction. For a specific value of wheel traction, the forces used to accelerate, brake, and turn the mouse must be correspondingly limited. Because of this, we try to optimize the performance of the mouse for a given force.

The force,  $F_{par}$ , which the motors, wheels, and ultimately the tires of the mouse can apply parallel to the floor of the maze to accelerate and move the mouse, is shown below.

$$F_{par} = \gamma M_m g = A_o M_m$$

$$A_o = \gamma g$$

where:

$\gamma$  = coefficient of friction of tires

$g$  = acceleration due to gravity

$A_o$  = acceleration of the mouse

The maximum acceleration,  $A$ , which the mouse can achieve is limited by the coefficient of friction of the tires and does not depend on the mass of the mouse. This assumes that the

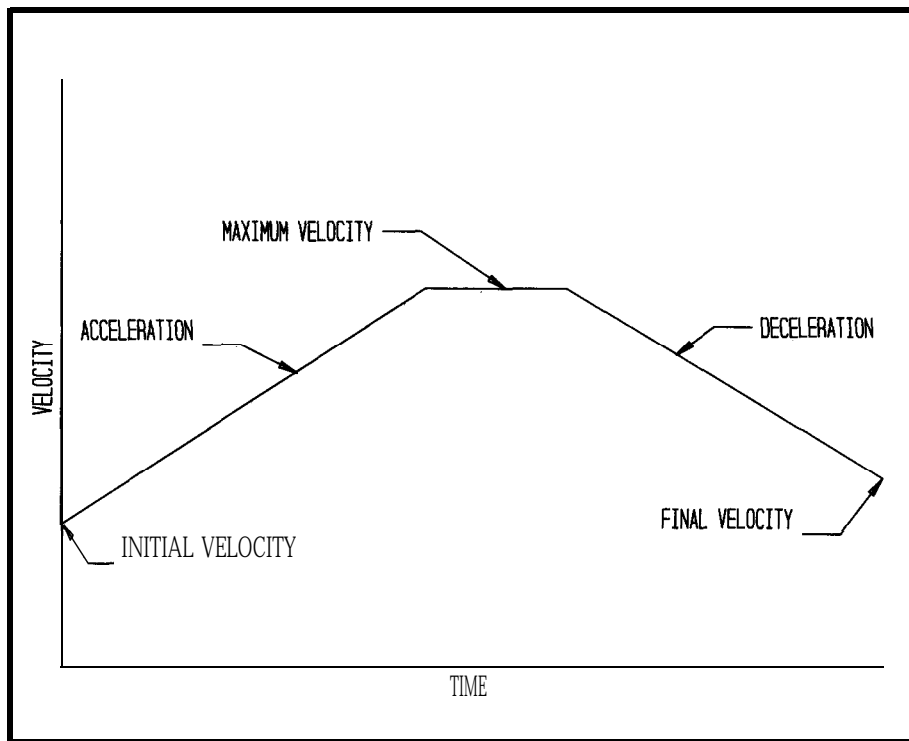
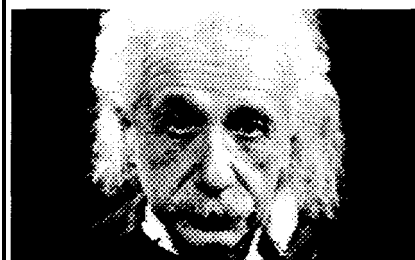


Figure 4-A **typical velocity profile along a straightaway is made up of five parameters: initial and final velocities, and maximum acceleration, velocity, and deceleration.**



### Introducing **VICTOR**, the video capture and image processing library

Victor is a library of functions for C programmers that simplifies development of scientific imaging, quality control, security, and image database software. Victor gives you device control, image processing, display, and TIFF/PCX file handling routines.

Your software can have features such as: live video on VGA, resize and zoom, display multiple images with text and graphics. Image processing functions include brightness, contrast, matrix convolution filters: sharpen, outline, etc, linearization, equalization, math and logic, overlay, resize, flip, invert, mirror. Size/number of images limited only by memory. Display on EGA/VGA up to 800x600x256. And, to get you up and running quickly, we've included our popular Zip Image Processing software for rapid testing and prototyping of image processing and display functions.

Victor supports Microsoft C, QuickC, and Turbo C... all for only \$195.

Victor and Zip support ImageWise and other popular video digitizers.

### NEW! **ZIP Colorkit**, the software that allows any gray scale digitizer to create photographic quality color images.

It's easy to produce stunning color images with the ZIP Colorkit -- capture three images with a gray scale video digitizer using red, green, and blue filters and save the images. Colorkit loads the image files and uses a unique optimizing algorithm to calculate the optimum color image. Supports PIW, PIF, PCX, TIFF, GIF, and TGA file formats. ZIP COLORKIT, \$75.

VICTOR LIBRARY includes FREE ZIP Image Processing ..... \$195

VICTOR LIBRARY with video frame grabber ..... \$349

ZIP Colorkit ..... \$75

**Call (314) 962-7833 to order**  
VISA/MC/COD

CATENARY SYSTEMS  
470 BELLEVIEW  
ST LOUIS MO 63119  
(314) 962-7833

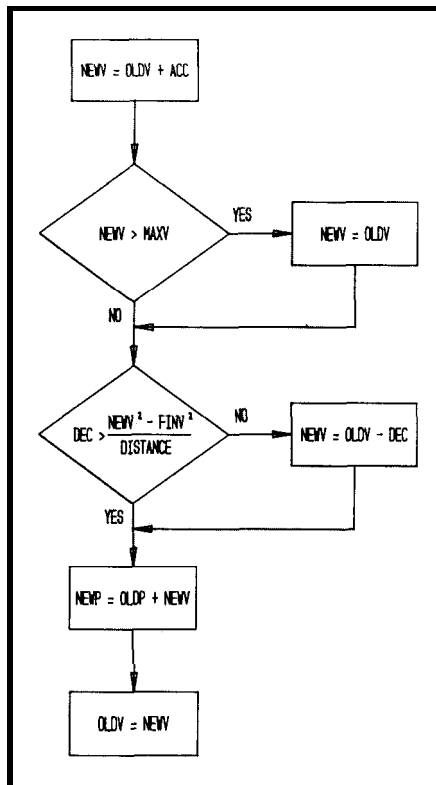
entire weight of the mouse is distributed on the wheels that are providing the acceleration. If part of the weight is on a passive wheel such as a caster, the maximum acceleration will be reduced accordingly.

To make life easy for MITEE Mouse III while it is running through the maze, it uses a system of units which are more natural for it than meters or seconds. The unit of distance is the pulse and corresponds to the distance the mouse must travel to register one pulse on the shaft encoder. The unit of time is 1 ms because that is the time between processor interrupts. Accordingly, one square is 4176 pulses, a 90° turn in place is 1260 pulses, velocity is **measured** in pulses/ms, and acceleration in **pulses/ms<sup>2</sup>**.

The motion control system of the mouse is concerned with three different quantities: position; its derivative, velocity; and its derivative, acceleration. Because the velocity is in the middle and the position or acceleration can be computed from the velocity with only one operation, the profile generator computes velocity profiles. Of course, the velocity profile must have a slope or derivative consistent with the desired acceleration, and the integral must be consistent with the desired position.

#### STRAIGHT PATHS

When traveling in a straight line, the maximum available force can be used to accelerate or brake the mouse. If the mouse starts and ends the straight with equal velocity, it can cover the distance in the least amount of time (for a given acceleration) by **accelerating** for half the distance and decelerating for the remaining distance. If it is moving at different velocities at the start and finish, the **acceleration** and deceleration periods are not equal, but the mouse will always be accelerating or braking. For some types of straights such as long diagonals, it may be desirable to limit the maximum velocity, not because that is the fastest way to travel, but because it is the safest. Thus the straight-profile generator creates a velocity profile **based on five** Parameters: **starting** velocity, **maximum** acceleration, **maxi-**



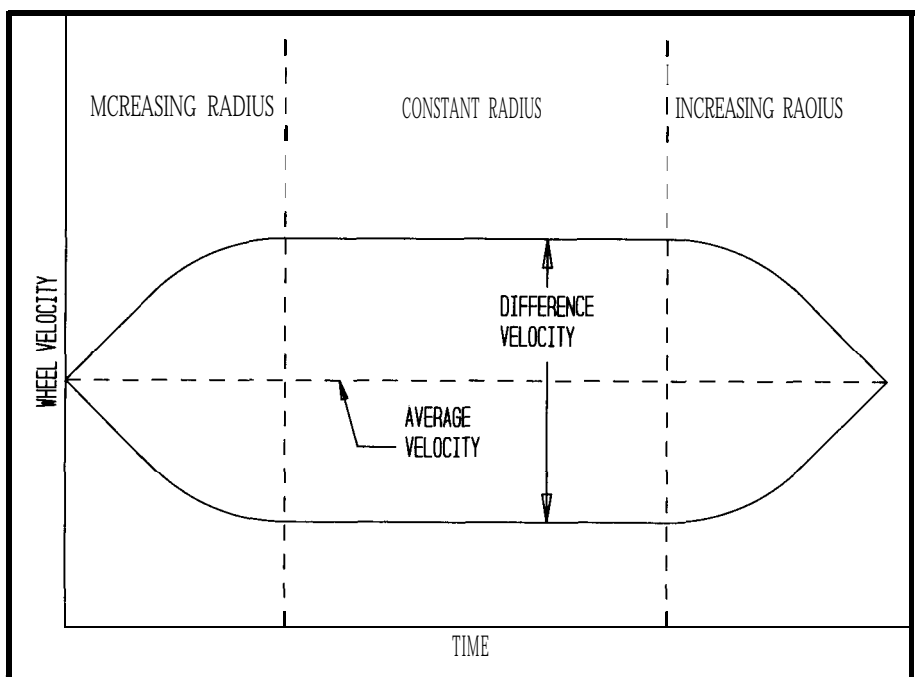
**Figure 5—** The straight-profile generator is responsible for controlling the mouse on straightaways.

imum velocity, maximum deceleration, and final velocity. A typical velocity profile is shown in Figure 4.

One important feature to have in the straight-profile generator is the ability to change the total distance in

real time. This requirement comes about because the mouse continually updates its current position based on signals from its sensors. At one instant of time it may think that it still has 20 cm to go before the end of the straight, but the sensors may see an opening in a wall signaling that it only has 18 cm to go. If the mouse was already decelerating at the maximum rate based on 20 cm remaining distance, it has a problem. If it continues to decelerate at the current rate, it will reach the end with too high a final velocity. If it tries to brake harder to reach the final velocity, it will exceed the **maximum** deceleration acceptable for its wheel traction and the wheel will begin to skid. MITEE Mouse III is programmed to brake harder to avoid entering the turn at **the end** of a straight with too high a velocity. If, however, it gets to the start of the turn and the velocity is still too high, it tries to go through the turn at the higher speed.

Figure 5 shows a flow chart of the straight-profile generator. Each time through the program, **NEWV** variables are calculated from **OLDV** ones. Variables ending in **V** are velocities, those ending in **P** are positions. **ACC** is the target acceleration, and **DEC** is the target deceleration. The variable **FINV<sup>2</sup>** is the final velocity squared



**Figure 6—** Turning corners is as simple as running the motors at different speeds, but knowing the speed differential to use for a particular corner can be tricky.

and is precalculated at the start of every straight so that only *NEW* must be calculated every millisecond. The velocity is stored in **pulses/ms**. This is too coarse, however, so **an extra** byte is used to the right of the decimal point, allowing resolution to **1/256** of a **pulse/ms**. The acceleration and deceleration is specified as **pulses/ms<sup>2</sup>/256**. This allows the new velocity to be calculated by simply adding the acceleration to the old velocity. The new position is calculated from the old position by **simply adding** the new velocity to it. The acceleration is stored in one byte, the velocity in two, and the position in three. Although the position is calculated to **1/256** of a pulse, only whole numbers of pulses are sent to the servo. The fractions are accumulated until they round off to a whole pulse. Because the profile generator only requires one unsigned multiplication and division, and mostly **16-bit** addition, subtraction, or comparison operations, it is efficient and runs in a very short time.

The straight-profile generator has information on the speed and accel-

eration of the mouse and therefore generates feed-forward information that is later used by the motor control routines. As mentioned before, the feed-forward terms in the control significantly decrease the error in the wheel position without increasing the gain or bandwidth of the feedback loop.

#### TURNS

There are two types of turns. During the search phase, the mouse turns in place. This is a very simple type of turn, and the profile generation is similar to a straight, except one wheel rotates forward and the other backward. The acceleration and deceleration intervals are equal, the initial and final velocities are zero, and the distance is always fixed.

To maximize the speed through the maze, however, it is desirable to avoid starting and stopping every straight from zero velocity. Also a smooth and continuous turn will require less time than a turn in place. As with the straight, to turn in the **short-**

est time, it is necessary to keep the forces on the tires at the maximum allowable value at all times. If the mouse were to travel at a constant speed around a constant radius turn, the only forces on the tires would be due to centrifugal forces. The average angular velocity,  $\omega_{avg}$ , of the wheels around such a turn would be:

$$\omega_{avg} = \frac{\sqrt{A_o R_t}}{R_w}$$

where:

$R_t$  = radius of turn

The inside wheel would turn at one constant speed and the outside wheel at a higher constant speed. The difference in wheel speed,  $\omega_{dif}$  would be:

$$\omega_{dif} = \frac{R_m}{R_w} \sqrt{\frac{A_o}{R_t}}$$

One minor difficulty is that the mouse comes into a turn from a straight where both wheels are at the same speed, but during the turn the wheels rotate at different speeds. A step change in speed, in this case a step

## EXPRESS CIRCUITS

MANUFACTURERS OF PROTOTYPE PRINTED CIRCUITS FROM YOUR **CAD** DESIGNS  
TURN AROUND TIMES AVAILABLE FROM 24 HRS — 2 WEEKS

#### Special Support For:

- TANGO. PCB
- TANGO SERIES II
- TANGO PLUS
- PROTEL AUTOTRAX
- PROTEL EASYTRAX
- smARTWORK
- HiWIRE-Plus
- EE DESIGNER I
- EE DESIGNER III
- PADS - PCB
- OTHER PACKAGES ARE NOW BEING ADDED
- FULL TIME MODEM
- GERBER PHOTO PLOTTING

*Express*  
Circuits

314 Cothren St., P.O. Box 58  
Wilkesboro, NC 28697

Quotes:  
1-800-426-5396  
Phone: (919) 667-2100  
Fax: (919) 667-0487

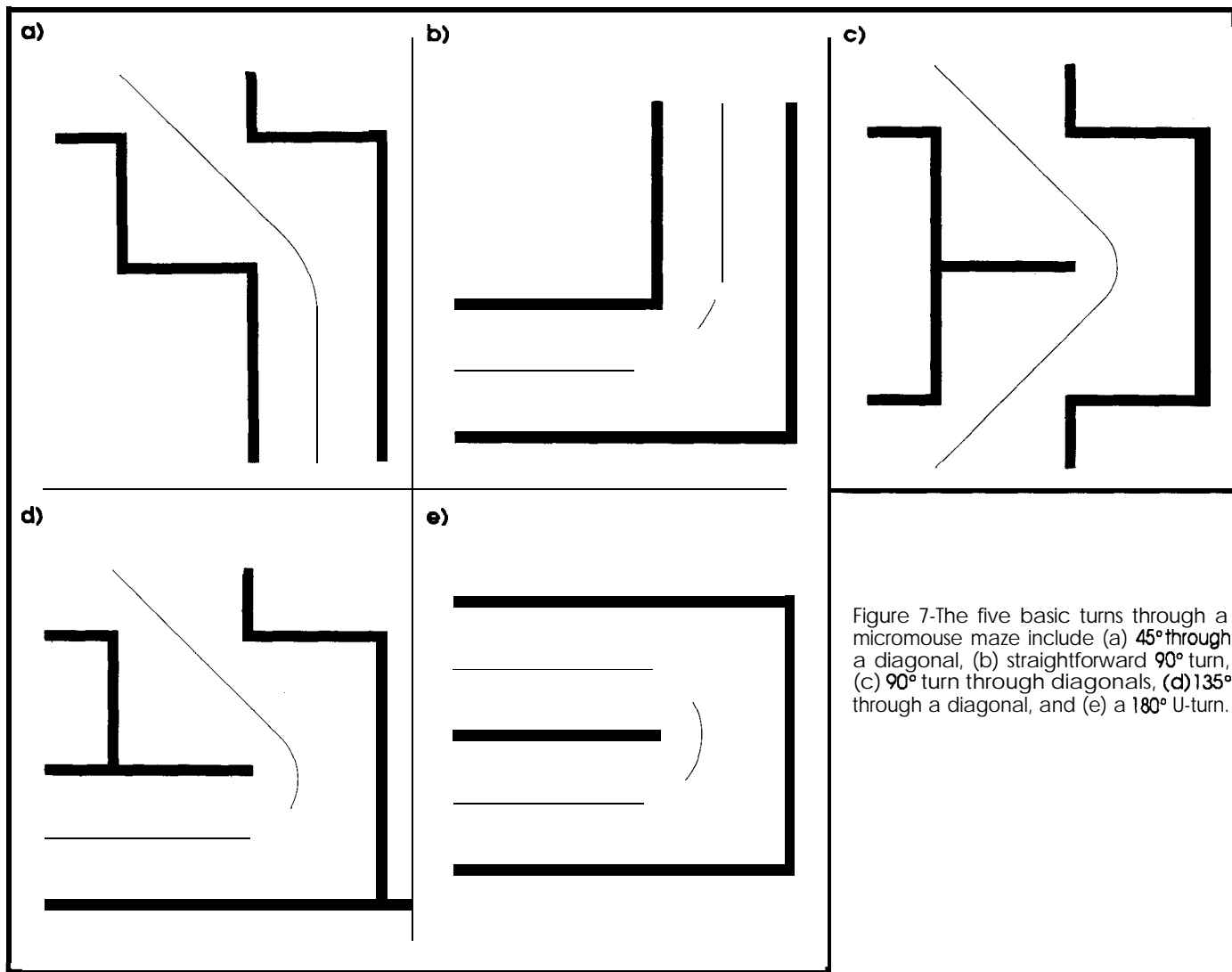


Figure 7-The five basic turns through a micromouse maze include (a) 45° through a diagonal, (b) straightforward 90° turn, (c) 90° turn through diagonals, (d) 135° through a diagonal, and (e) a 180° U-turn.

change in the differential speed, would take an infinite force from the tires to accelerate the inertia of the mouse in zero time. Figure 6 shows the actual velocity of each wheel as it goes around a corner. It turns out that the net force on the tires is constant if each wheel accelerates up to the target differential velocity with a time-dependent profile as shown:

$$\sin\left(2R_m \frac{M_m}{J_m} \sqrt{A_o R_t} t\right)$$

where:

$t$  = time

Based on this equation, a mouse should be designed with widely separated drive wheels and a large mass-to-inertia ratio. Unfortunately MITEE Mouse III has the drive wheels spaced close together so that it can navigate diagonals. This clearly compromises

its turning performance. The sensors, however, are supported with a balsa wood frame to minimize their contribution to the inertia of the mouse.

During the time the mouse is accelerating to full differential velocity it is going on a circle with a constantly diminishing radius. To determine the distance the mouse travels during this phase, the  $x$  and  $y$  components of the velocity were numerically integrated using MathCAD. This information is then stored in a table and used by the turn-profile generator to make sure that each turn is initiated at the correct position to complete the turn in the center of the square and heading in the correct direction.

There are five types of turns required by MITEE Mouse III when making its high-speed runs through the maze. These are shown in Figures 7a to 7e. Each of these has a different

radius and therefore different average and difference velocities. The straight-profile generator checks to see the type of turn required at the end of each straightaway and sets the final velocity of the straight equal to the average velocity for that type of turn. The turn-profile generator then uses a look-up table to find the other parameters of the turn and generate the appropriate turn profile.

As with the straight-profile generator, the turn-profile generator has information on the speed and acceleration of each wheel and therefore generates feed-forward information that is later used by the motor control routines.

## NAVIGATION

Perhaps the most difficult part of building a micromouse is the naviga-

tion. It is often left until last and cannot be tested until the other systems are operational, but it determines to a great extent the capability and reliability of the mouse. For humans, navigation comes naturally. By collecting information with our eyes and recognizing what we see, we can walk, run, or drive a car in a straight line or around a corner without much difficulty. A micromouse, by contrast, does not do so well. For its size it goes quite fast. MITEE Mouse III may hit 8 MPH on a 15-square straight. If we scale its speed according to its size, this is equivalent to driving a car down a road at almost 200 MPH, looking out the side window and trying to keep the car exactly 3 feet from the curb.

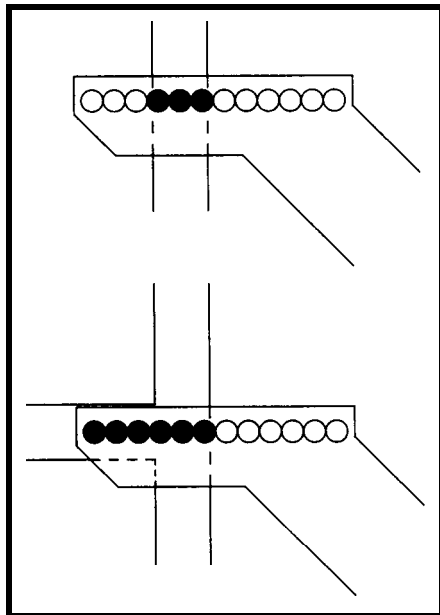
The first type of navigation most mice employ is to measure position from their wheels. By knowing the wheel diameter and measuring the number of wheel rotations, a good estimate of position can be maintained. This works very well at low speed when the tires are not slipping. MITEE Mouse III can go 10-15 squares without any other correction if it is properly aligned at the start. What it cannot do, however, is search the whole maze and then do a speed run without additional corrections.

The eyes of the navigation system are the sensors. On MITEE Mouse III, they are made up of twelve focused infrared emitter-detector pairs located in linear arrays at the four corners of the mouse. If a wall or post is located directly under an emitter-detector pair, light from the emitter will reflect back from the surface and turn on the detector. If only the floor is directly under an emitter-detector pair, the detector will remain off. Both the tops of the walls and the tops of the posts look the same to the sensors. Figure 8 shows the detectors that are turned on by different wall conditions.

Once the detector signals are received by the processor, they are converted into a distance or clearance. This clearance is measured in units of emitter-detector pair spacings (0.2 inches) and represents the position of the detector, closest to the mouse, that is turned on. A clearance of 12 means that all the detectors are off and there

is no wall under any part of the array. A clearance of 0 means that at least the detector nearest to the mouse is on.

There are three basic types of errors the mouse must recognize and correct. These are called forward error, heading error, and offset error. The first of these, forward error, is illustrated in Figure 9a. It is usually detected by openings in the walls on either side of the mouse. When there is no wall or post under a sensor array, none of the detectors are turned on. When any of the detectors first turns on, it defines the leading edge of a wall. The trailing edge is defined when the mouse passes over a wall and the last detector is turned off. The position of the wheels is recorded at each leading and trailing edge.



**Figure 8—Mouse-to-wall distance can be determined by how many sensors are active. Intersections can also be detected.**

The rules of the contest specify that there must be a post at the four corners of every square. The sensors for MITEE Mouse III are designed such that the four posts will be directly under the four sensor arrays when the mouse is in the center of a square. As the mouse enters a square, the sensors should detect a leading edge when the mouse is half a post width from the center and detect a trailing edge when the mouse is half a post width past the center. Any difference between the expected and measured leading and trailing edges is an error. The meas-

# 8031/51 Tools

## 8031 In-Circuit Emulation

Our emulator provides most features of an 8031 In-Circuit-Emulator at a significantly lower price. It assists in integration, debug, and test phases of development. Commands include: disassembly, trace, breakpoint, alter register/memory, and load Intel Hex file. **\$199**

## 8051 Simulation

The 8051 SIM software package speeds the development of 8051 family programs by allowing execution and debug without a target system. The 8051 Simulator is a screen oriented, menu command driven program doubling as a great learning tool. **\$99.**

## 8031/51 Single Board Computer

A fast and inexpensive way to implement an embedded controller. 8031132 processor, 8+ parallel I/O, up to 2 RS232 serial ports, +5 volt operation. The development board option allows simple debugging of 8031/51 family programs. **\$99ea**

## Prototyping System

The IPC-52 development system allows you, the designer, to concentrate on Application Specific Circuitry only, because the 8052, RAM, EPROM, and I/O sections are built-in and fully functional. The prototyping bread-board is integrated into the system - save days of development time. **\$220**

## Call us for your custom product needs.

Other products available:

**MyGAL - GAL Programmer \$199**

**FORTH Card - FORTH development card for STD Bus \$279 (OEM-\$199)**

**HTE**

HiTech Equipment Corp  
9400 Activity Road  
San Diego, CA 92126  
[FAX: (619) 530-1458]

**(619) 566-1892**

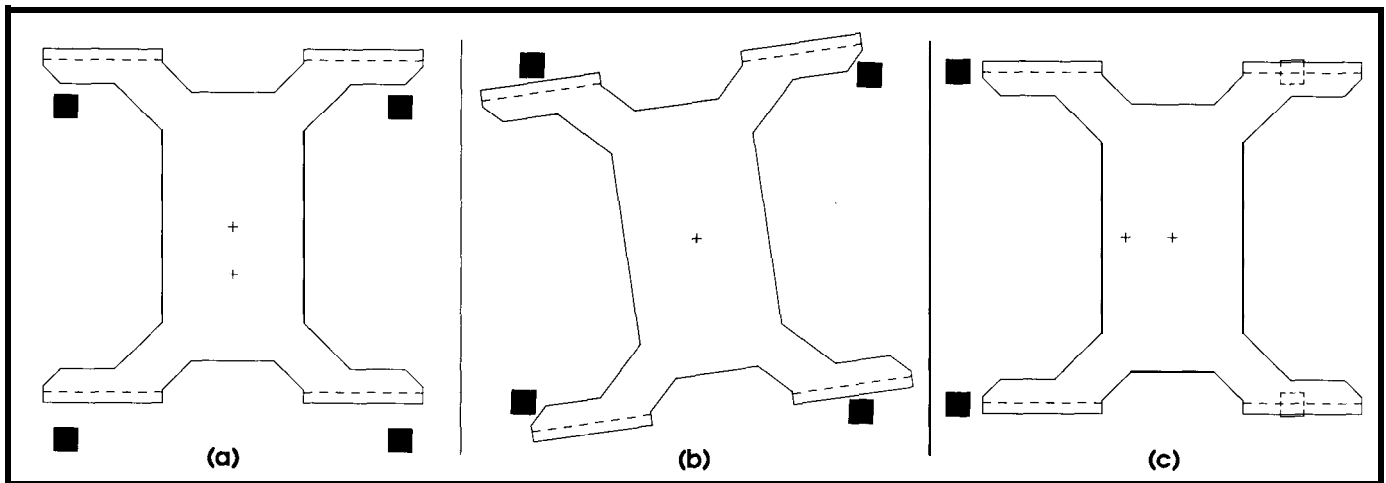


Figure 9—MITEE Mouse III recognizes three types of positional errors including (a) forward error, (b) heading error, and (c) offset error. The squares represent the four corners of the maze square and the crosses represent ideal position and actual position.

urements from the four sensor arrays are averaged to correct the forward error. To make sure that no edges are missed, MITEE Mouse III is programmed to start looking for edges 4 cm before it reaches the center of a square, and continue looking until 4 cm after it leaves the center.

The forward error is the **difference** between where the mouse thinks it is and where it actually is. This can be corrected by updating where the mouse thinks it is. Since this does not involve any mechanical motion, it can be done instantaneously.

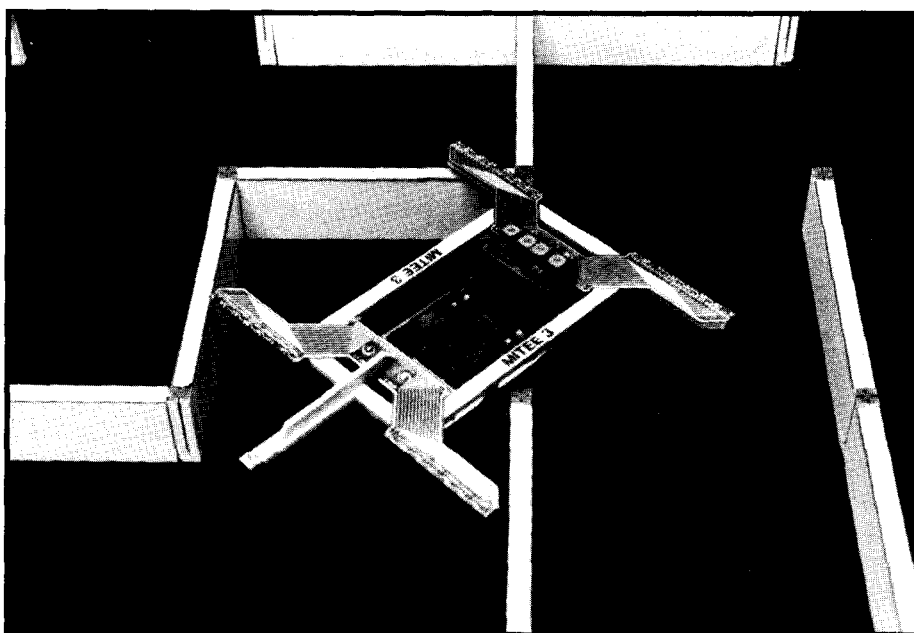
If the sensors were perfect, they would detect a wall only when it was located directly below them. The infrared emitters and detectors use lenses **which** collimate and focus the light, but the sensors still detect the wall slightly before and after it is directly under them. This gives an effective wall width, as seen by the sensors, which is slightly greater than the actual width. This is not a problem because it can be compensated for by software. A problem does

arise, however, when some sensors are more sensitive than others and pickup the wall sooner and see it later. Also, in spite of circuitry specifically designed to suppress ambient light, there are some second-order effects which change the sensitivity of the detectors due to ambient light. Lastly, the sensitivity **is** affected by the height of the sensor above the walls. Both rocking of the mouse on its wheels and variations in the fabrication of the maze aggravate this problem. To keep the software simple, a single value of effective wall width is used, even though this guarantees an error with some sensors. This error is mitigated, however, because in many cases the sensor which first picked up a wall

will also be the last to see it. If the positions of the leading and trailing edge are averaged, the width of the wall drops out and the true center is correctly determined.

On a long straight section there may not be any openings in the side walls. Under these circumstances there may be no leading or trailing edges from which to correct the forward error until the very last square where the mouse must turn left or right. The mouse must rely on its tire measurements. Unfortunately, long straight sections are where the mouse reaches the highest velocity and the tire measurements are the least accurate. In addition, the maze is often made of sections or quadrants and these quadrants

join in the middle of the longest straights. It is often difficult to adjust the quadrants so that the joints are flat. As a result, the mouse bounces up when it hits the joint, loses track of exactly where it is, and cannot correct the error until it is almost too late. A carefully adjusted **maze** is the best solution to this problem, but if the maze is





warped, it may not be possible to adjust it for low error over the full length of each piece. To compensate for this, MITEE Mouse III is programmed to perform its first speed run with a low maximum velocity in case there are any bad bumps. The maximum velocity is then raised for all subsequent runs, so that if the maze is flat, the mouse will realize its best possible time.

The second type of error, heading error, is illustrated in Figure 9b. It represents a mistake in the direction the mouse is traveling. It is very serious because the mouse will crash if it is not corrected. The heading error is easy to correct, however. If the mouse is built with a wheel chair design, such as MITEE Mouse III, one wheel must simply turn faster for a period of time and the other turn proportionally slower. If the mouse is standing still, the heading can still be corrected. One wheel must turn forward a fraction of a turn and the other turn backward by the same amount. The mouse will rotate in place and the heading will be corrected.

The third type of error, offset error, is illustrated in Figure 9c. When a mouse is traveling down a straight section, but is not in the center of the track, it has an offset error. Offset error is less serious than heading error because an offset error alone will not cause the mouse to crash. It is more difficult to correct, however, because the mouse is not able to translate directly to the side. To correct an offset error the mouse must first generate a heading error, travel with that heading error until the offset error is corrected, and then correct the heading error.

Heading and offset errors are often dealt with together because the same set of measurements are used to calculate both. During the time the mouse is looking for leading and trailing edges, it also keeps track of the minimum clearance on each of the four sensor arrays. These measurements are used to calculate the heading and offset errors.

$$\text{heading error} = \frac{LFC - RFC - LBC + RBC}{2}$$

$$\text{offset error} = \frac{LFC - RFC + LBC - RBC}{2}$$

where:

- LFC** = left front clearance
- RFC** = right front clearance
- LBC** = left back clearance
- RBC** = right back clearance

In contrast to forward errors, both of these errors cannot be corrected instantaneously. The inertia of the mouse limits how quickly the heading can be corrected. MITEE Mouse III tries to correct the heading before it travels an appreciable part of one square. It uses an acceleration for the heading correc-

tion which is proportional to the velocity it is traveling. When it is going slowly, it corrects slowly; when it is moving fast, it corrects quickly. The offset error has all the limitations of the heading error and, in addition, distance is required for the offset to be corrected. MITEE Mouse III is programmed to correct the offset error in exactly one square. It generates a heading error that will exactly cancel the offset error by the time the next clearance measurements are made. Because the clearance measurements are made from the posts and not the walls, new heading and offset errors

## GET TO WORK!

### A New Project

Our line of macro Cross-assemblers are easy to use and full featured, including conditional assembly and unlimited include files.

### Get It To Market--FAST

Don't wait until the hardware is finished to debug your software. Our Simulators can test your program logic before the hardware is built.

### No Source!

A minor glitch has shown up in the firmware, and you can't find the original source program. Our line of disassemblers can help you re-create the original assembly language source.

### Set To Go

Buy our developer package and the next time your boss says "get to work", you'll be ready for anything.

### Quality Solutions

PseudoCorp has been providing quality solutions for microprocessor problems since 1985.

### BROAD RANGE OF SUPPORT

- Currently we support the following microprocessor families (with more in development):

Intel 8048	RCA 1802,05	Intel 8051	Intel 8096
Motorola 6800	Motorola 6801	Motorola 68HC11	Motorola 6805
Hitachi 6301	Motorola 6809	MOS Tech 6502	WDC 65C02
Rockwell 65C02	Intel 8080,85	Zilog Z80	NSC 800
Hitachi HD64180	Motorola 68000,8	Motorola 68010	

- All products require an IBM PC or compatible.

**Cross-Assemblers** as low as \$50.00

**Simulators** as low as \$100.00

**Cross-Disassemblers** as low as \$100.00

**Developer Packages** as low as \$200.00

(a \$50.00 Savings)

**So What Are You Waiting For?**

Call us

**PseudoCorp**

**Professional Development Products Group**

716 Thimble Shoals Blvd, Suite E

Newport News, VA 23606

**(804) 873-1947**

## MICON-196KC

### DIGITAL CONTROLLER KIT



The MICON-196KC Digital Controller can be used in applications such as: high speed closed loop motion control, mid-range digital signal processing, high speed positioning control, traction control systems and anti-lock brakes, fast robots, high speed data acquisition, etc.

With a foot print of 3.5x3.5", the MICON-196KC Kit consists of: a CPU Module featuring the 80C196KC, Intel's 16 bit/16 MHz embedded controller; a MEMORY Module containing 64K EPROM/RAM with customized memory mapping; a PROTO Module supplied for customer's application specific circuitry; two BUS Modules, a Firmware/Software support package, and a User's Manual.

The MICON Kit can be configured in at least 5 different modes: as a low-cost software development tool in conjunction with any IBM PC-AT or compatible; as a learning tool using application-oriented tutorial software; as a hardware prototyping environment for customized I/O; as an EPROM burner for the 87C196KC microprocessor; or as a READY-TO-USE digital control solution for your applications.

The MICON-196KC is aimed to restructure the relationship between HARDWARE/SOFTWARE/APPLICATION engineers, and to build Inter-Disciplinary bridges. This product is OEM-Ready and provides a cost-effective way to start BIG PROJECTS with SMALL BUDGETS!!!

MICON-196KC sells for \$299.00

(Additional Hardware and/or Software Enhancement Kits for MICON-196KC are also available.)

## MICON

5270 Elvira Road, Bldg. 104  
Woodland Hills, CA 91364  
Phone (818) 348-4992  
Fax (818) 348-0960

Intel is a Registered Trade Mark for Intel Corp.  
IBM PC-AT is a Registered Trade Mark for IBM Corp.

Reader Service #151

can be calculated and corrected every square regardless of the maze design.

Though heading and offset errors are calculated from the same measurements, it is important to separate these two types of errors, particularly before a 90° turn. This is due to the fact that a heading error will still be a heading error after a turn, but an offset error will become a forward error. Also any forward error will become an offset error. Forward errors are generally well corrected before entering a turn, so there should be no offset error after the turn. Offset errors

by contrast, cannot be corrected if they are detected just before a turn. They should not be a problem, however, because they will become forward errors after the turn and forward errors are easy to correct. With this in mind, MITEE Mouse III is programmed to ignore offset errors if the next move is a turn.

## DIAGONAL NAVIGATION

MITEE Mouse III requires an additional type of navigation not required on most mice: the ability to navigate diagonals. Since the mouse is not running parallel to any walls, this may appear difficult at first, but it turns out to be quite simple. The program for diagonal navigation is less than half the length of the program for normal navigation.

The same type of error detection and correction—forward, heading, and offset—are required for diagonals as for straights. The difference is in how the required data is collected. Figure 10 shows MITEE Mouse III running down a diagonal. The left front sensor array is starting to pick up a corner. As the mouse moves forward, the clearance will get smaller and then larger. As with the straight navigation, the minimum clearance is recorded and stored. The wheel position when the clearance was at a minimum is also saved. As the mouse

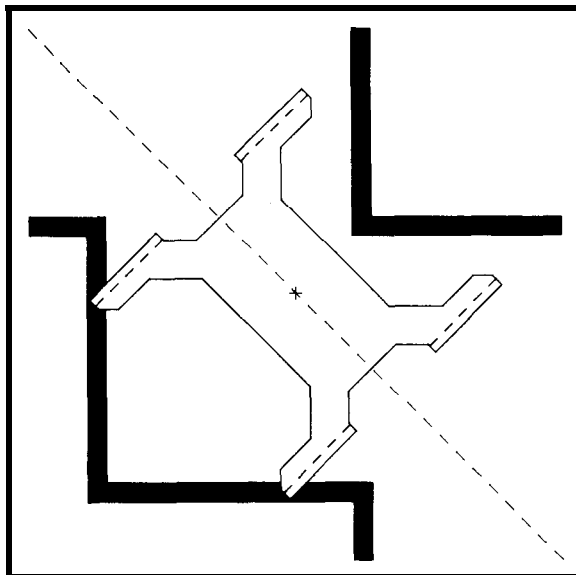


Figure 10—MITEE Mouse III has the unique ability to run diagonally, so must be able to collect data while doing so.

moves forward the right rear sensor array will start to pick up a second corner. The minimum clearance and wheel position at minimum clearance is also recorded and stored. The two wheel position measurements are used just like the leading and trailing edge measurements to correct the forward position. The minimum front and rear clearances are used to calculate the heading and offset errors.

$$\begin{aligned} \text{heading error} &= 2C_d - MFC - MBC \\ \text{offset error} &= MBC - MFC \end{aligned}$$

where:

$$\begin{aligned} C_d &= \text{clearance on a diagonal} \\ MFC &= \text{minimum front clearance} \\ MBC &= \text{minimum back clearance} \end{aligned}$$

If the left front and right rear sensors are used to make the clearance measurements for one square, the adjacent square will use the right front and left rear sensors. The corrections work very well but are less precise than those for straight navigation for several reasons. First, they are based on two measurements instead of four, resulting in more noise in the measurements. Secondly, the total clearance on each side of the mouse is only 0.6 inches. The resolution of the sensors is one emitter-detector pair or 0.2 inches. A resolution of one third of the clearance does not allow precise correction.

The third reason is more subtle, and is due to the placement of the sensors. They are not optimized for diagonal operation and cannot simultaneously measure the position of a corner in front of the mouse and behind it. The front corner is measured first and a short distance later the back corner is measured. We assume that the mouse travels in a straight line between the two measurements. The resulting heading and offset measurement was valid when the mouse was centered between the two corners but may not be valid now that the mouse has traveled farther. In fact, if the mouse had a heading error when it was centered between the corners, it has been traveling on an incorrect heading since then, and has an offset error by the time the measurement is finished. If we know the distance that the mouse has traveled since it was centered, we can estimate the resulting offset error and add that to the offset error measured when the mouse was centered. The heading error remains the same as the value when it

was centered. If the corrections are then performed on the combined offset error and original heading error, the stability of the corrections, particularly for large errors, is significantly improved.

Nevertheless the maximum velocity of the mouse is usually limited on diagonals to give the mouse time to correct each error as it is detected. The time saved by going directly down a diagonal more than offsets the loss of time due to limiting the maximum velocity.

### ..AND ONWARD

Armed with the information in this article and the related computer programs available through the Circuit Cellar BBS-information which I have spent the last five years putting together-a newcomer to the micro-mouse arena will hopefully become competitive in a much shorter time. Accordingly, I hope to see the day when MITEE Mouse III clones are beating the original and making a good

account of themselves on the world scene. To those of you who are not interested in micromice but insist on applying all this high-performance stuff to some ungainly robot, slowly ambling toward the outlet in their living room-it's not what I had in mind, but if it is helpful, great!+

I would like to acknowledge the help of Tony Caloggero who built all the mechanical components for MITEE Mouse III, Andy Goldberg who worked on the maze solver, even when his doctoral thesis was on the line, and Leo Casey who was my constant sounding board and helped with the navigation. Without their help, there would have been no mouseabout which to write an article.

David Otten is a principle research engineer in the Laboratory for Electromagnetic and Electronic Systems at the Massachusetts Institute of Technology. He holds a B.S. and M.S. degree from MIT.

### IRS

- 2 10 Very Useful
- 2 11 Moderately Useful
- 2 12 Not Useful

## MS-DOS EPROM PROGRAMMING SYSTEM NEEDS NO INTERNAL CARD

**EPROMS**  
2708 (3 supply)  
2758, 2716  
27C16, 2516  
2532\*, 2564\*  
68764\*, 68766\*  
2732, 2732A  
27C32, 2764  
2764A, 27C64  
27128, 27128A  
27C128, 27256  
27C256, 27512  
27C512, 27C010\*  
27010\*, 27C1001\*



**EEPROMS**  
2804, 2816A  
2864A, 28256\*

**MicroControllers**  
8741A\*, 8742\*  
8748\*, 8748H\*  
8749\*, 8749H\*  
8751\*, 87C51\*  
8752\*, 8744H\*

\*Socket Adapter Required (Diagrams Included)

CONNECTS TO YOUR SYSTEMS

### PARALLEL PRINTER INTERFACE

A FULL FEATURED, EASY-TO-USE SYSTEM WORKS WITH ANY DESKTOP OR LAPTOP MACHINE ADAPTIVE, HIGH-SPEED ALGORITHM MINIMIZES PROGRAMMING TIME AND INSURES VALID DATA SYSTEM PROGRAMS ALL STANDARD DEVICES OR EQUIVALENTS FROM ANY MANUFACTURER ALL SYSTEM COMPONENTS FIT NEATLY INTO CASE FOR TRAVEL OR STORAGE

### SYSTEM SOFTWARE COMMANDS

PROGRAM EPROM(S)	SAVE EPROM(S) OR	COPY EPROM(S)
FROM DISK FILE	BUFFER TO DISK	VERIFY EPROM
READ DISK FILE INTO	PROGRAM EPROM(S)	ERASED
BUFFER	FROM BUFFER	BUFFER EDITOR
READ EPROM(S) INTO	COMPARE EPROM(S)	SELECT DEVICE TYPE
BUFFER	WITH BUFFER	DEVICE CHECKSUM

BUFFER EDITOR HAS 18 BYTE LEVEL COMMANDS FOR DETAILED OPERATIONS

SYSTEM INCLUDES: PROGRAMMING UNIT, POWER PACK, CONNECTING CABLE, OPERATION MANUAL & SOFTWARE **\$239**

SOFTWARE AVAILABLE ON 3 1/2" OR 5 1/4" DISK  
TO ORDER SEND CHECK, MONEY ORDER, WRITE OR CALL

VISA

**ANDRATECH**  
P.O. BOX 222  
MILFORD, OHIO 45150  
(513) 831-9708

MASTER CARD

CALL OR WRITE FOR MORE INFO. - ADD \$5.00 FOR SHIPPING - 14.00 COD

New, Improved!

# SIBEC-II

The ideal solution for embedded control applications and stand-alone development.



- Intel 8052AH BASIC CPU
- Serial printer output and 5, 8 bit I/O ports
- 5 in.<sup>2</sup> prototyping area
- Memory: 8K RAM, expandable to 128K
- Power requirements: 5V.DC @ 300 ma. only
- PROM programmer; ZIF socket for 2764 or 27128 EPROM
- Interrupt handling capability
- Built to exacting standards and warranted
- Still only \$228.00 including documentation (quantity 1)

Inquire about our **PDK51** 8051-8052 product development kit for the IBM-PC/XT/AT: **\$595.**  
Our **BXC51** 8051/8052 BASIC compiler: **\$295.**

**Call now! 603-469-3232**

**Binary Technology, Inc.**  
Main Street . PO Box 67 . Meriden, NH 03770

Reader Service # 103

Reader Service #108

# Huge Arrays on the HD64180

## Taking Advantage of Memory Management

Jack Ganssle

Hitachi's HD64180 (and its little microcontroller brother, the HD647180X) are sometimes used in applications where huge quantities of data are collected or analyzed. Their CMOS low-power design makes them perfect for battery-powered data acquisition equipment. Or, as part of an instrument, their architecture is ideal for storing large amounts of data during analysis.

As I mentioned in a previous article, these processors include a powerful Memory Management Unit (MMU) designed to extend the address space to a full megabyte. With some understanding of the intricacies of the MMU, it's not too hard to upgrade an old Z80 application to make use of the extra memory. Of course, the HD64180 still uses a 64K logical address (for compatibility with the Z80), so a Z80 program doesn't suddenly get a nice 1-MB linear addressing range. The software must be somewhat restructured if huge programs or data areas are needed.

(As a refresher, "logical" addresses are those issued by the program; on the HD64180 these can never exceed 64K, the limit imposed by using 16-bit addresses in load and jump instructions. "Physical" addresses are those appearing on the CPU's pins; the internal Memory Management Unit converts Logical to Physical, using a translation algorithm controlled by the programmer.)

Suppose you're designing a remote data collection device that gets one 16-bit ADC reading every second. This doesn't sound like much data, but after only a day the system will have acquired 86,400 words (172,800

bytes&—far more than the logical address space of 64K.

**Managing an HD64180's memory** in this sort of application demands careful analysis of both the logical and physical address space needs. There is no way all 176K will fit within the 64K logical space, so the program will have to remap the MMU, possibly often, to get to the array.

### ACCESSING ARRAYS

This problem requires "sequential" access to the data, since the current data point is appended immediately after the last. It is a special case of the more general array accessing situation, where a program might need any arbitrary data point, in no particular order.

Byte-oriented data (like strings) is easy to work with. Element *I* is the "Ith" address in the string; that is, the address of DATA(8) is the start address of the array plus 8 (assuming DATA(0) is a valid element). This is called a vector—it has only one dimension: that given by the subscript.

Of course, in real applications we often work with data that is 16 or more bits long. Integers are often two bytes; floating-point values typically are four or even eight bytes. Array element *I* of a vector is found from:

$$\text{base} + I \times (\text{element size})$$

where *base* is the start address of the array.

Again, this assumes element 0 is valid. Element 0 is at the first address of the array, followed sequentially by each other element.

Multidimensional arrays use an extension of this formula. In most systems these arrays are stored in "row major" order: given A(row, column), all column elements of row *O* are stored first, then the column elements of row 1, and so on. We can get to any element A(I, J) using the formula:

$$\text{address} = \text{base} + I \times (J_{\text{max}} \times E_{\text{size}}) + J \times E_{\text{size}}$$

*J<sub>max</sub>* is the number of columns in the array and *E<sub>size</sub>* is the number of bytes per element of the array.

As you can imagine, this can be a computationally expensive way to get to an array. Multiplications are slow. The HD64180's multiply instruction only handles 8-bit operations, and so by itself it is not adequate for indexing into an array. Generally you can count on the quantity (*J<sub>max</sub>* × *E<sub>size</sub>*) to be a constant; for speed-critical applications, it may be wise to set this to a convenient value (a power of two), even if some memory is wasted. Similarly, aim for an element size that is 1, 2, 4, or 8 so shifts can be substituted for multiplies.

Any number of dimensions can be accommodated by extending the formula. Higher dimension arrays require even more math to access, so try to limit the dimensions to one or two.

In real-time applications it's often nice to support two forms of data access. A perfectly general form is useful for off-line data reduction; your application program can request any array element in any sequence. During data acquisition you might need a shortcut to avoid the computational

overhead of computing an array index. If data is gathered in some sequential form, it's easy to visualize the data as a one-dimensional vector (instead of a multidimensional array), and store each value sequentially. We'll explore both methods.

## THE MEMORY MANAGER

Sure, it's easy to write code to compute an array index along the lines presented. We'll run into trouble when the array gets too big. Suppose your code uses most of a 27236 EPROM, leaving only 32K of logical address space for data. If all of this were dedicated to a two-dimensional array consisting of 4-byte-long elements, then only 8192 elements fit in memory—**ARRAY(100,100)** exceeds address 64K.

Here the MMU comes to the rescue, but not without a lot of help from the programmer. Obviously, we can simply reprogram the MMU's control registers every so often and bank switch portions of RAM into the processor's address space. The secret to success is careful planning.

If you've never really blown a software project by immediately jumping into coding, then you are probably sick of hearing about software methodologies. In fact, as processors and programs get more complicated, careful design is far more important than it once was. Oh for the days of 4K programs! We could crank out a few thousand lines of code in a couple of weeks and be done. Now, when 300K+ programs are the norm, a carelessly designed system will be a disaster. Guaranteed.

This is certainly true when using any sort of memory manager. One penalty of the MMU is a segmentation of your logical address space that must be designed in up front, and that very likely can never be changed without completely rethinking the entire structure of the program.

Using huge arrays forces you into a three-bank memory model on the HD64180 (note that on other chips other options exist—e.g., the Z280's fantastically complex and powerful MMU will let you have up to 16 banks

in the logical address space). One bank points to the ROMed program; in most cases this logical-to-physical mapping will never change. Another bank accesses an area of RAM for program variables and transients. In all but extreme cases this will never be remapped, because the stack will reside here. Finally, one bank points to the huge array(s).

Figure 1 shows one possible configuration of the memory management unit. This assumes 32K of ROM from logical address 0000 to 7FFF, 28K of RAM from 8000 to EFFF, and 4K of huge array RAM at the end of memory.

Wait a minute—4K of RAM for huge arrays? That doesn't seem like much! This 4K section of the system's address space is essentially a

“window” into the huge array. The window is for storage of huge arrays. Never, never store the stack or other variables here, since remapping will invalidate the data. We have to provide a section of logical space to allow access to the data; the 4K window is this section. This is much like a disk buffer used in operating systems—data is written to the buffer and then flushed to disk when full. Its size is a result of the mapping resolution of the MMU—4K is the minimum amount we can allocate. You can always make this larger, which will cut down on MMU remapping, but you'll sacrifice either variable or program area.

The idea behind using this window is to compute the physical (20-bit) address of the proper array element, and then to position the

```

;
; Put an element in a huge 1-dimensional array.
; Here we assume:
;
; The "window" is from F000 to FFFF.
; BASE_CBR is the first (lowest) CBR value
; Register B has the data to store
; Register HL has the array index
;
; cbr = base_cbr + (index AND ff000) / 4096
; logical address = f000 + (index AND 0fff)
;
putld:
    ld    a,h           ; get high order index value
    and  a,0f0h        ; mask off upper 4 bits (cbr bits)
    rra                ; shift right 4 bits
    rra                ; {since we ignore lower 8 bits,
    rra                ; this is a 12-bit shift)
    rra                ; a - (index AND ff000)/4096
    add  a,base_cbr
    out0 (cbr),a       ; set new cbr value
    ld   (save_cbr),a  ; save cbr for nextld
    ld   a,h           ; high part of index
    or   a,0f0h       ; mask off high nibble & or in f0
    ld   h,a          ; hl is right logical addr in wind
    ld   (hl),b       ; save data
    ld   (save_log),hl ; save logical addr for nextld
    ret

;
; put register B in the huge dimensional array at the next
; open location. This assumes that putld was once called to
; index a specific value.
;
; On entry, B is the data to store
; save_cbr was set by putld or nextld
; save_log was set by putld or nextld
;
nextld:
    ld   hl,(save_log) ; last logical address used
    ld   a,(save_cbr)  ; last cbr used
    inc  hl            ; skip to next element
    jr   nz,nextld    ; jp if not exceeded the window
    ld   hl,0f000h     ; reinit to start of the window
    inc  a            ; next cbr value
    ld   (save_cbr),a  ; save current cbr
    ld   (save_log),hl ; save current logical address
    out0 (cbr),a       ; set cbr
    ld   (hl),b       ; save data
    ret

```

Ming 1—Sample code shows how the HD64180's memory management unit simplifies accessing large blocks of data.

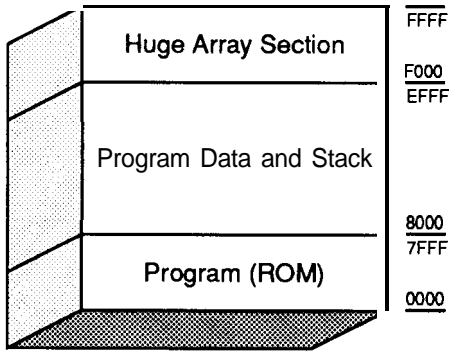


Figure 1 -One possible MMU configuration has 32K of ROM from 0000 to 7FFF, 28K of RAM from 8000 to FFFF, and a 4K array window from F000 to FFFF.

window to bring the data into view. Earlier we saw how to compute an index into any array. Now, this windowing step is also introduced. The algorithm is not complex, but a wise programmer will insulate his code from the machinations of array element lookup by hiding the details in subroutines.

The routine **PUT1D** stores a byte to a huge one-dimensional array using the map shown in Figure 1. The code to get a byte from the array is nearly identical and so is not shown. **PUT1D** accepts an array index in HL that can range all the way up to 64K. Thus, the one 4K window in logical address space gives the routine access to a 64K hunk of data. Note that it supports random accesses-it stores the data in B into the array at the index in HL. HL can be 12 on one invocation and 40000 on the next.

**PUT1D** locates the array in physical memory at address:  $0F000 + \text{BASE\_CBR} * 4096$ . Thus, if **BASE\_CBR** is 1, then the array runs from 10000 to 1FFFF. This implies that the memory manager's CBR register must be set to Fx (where "x" defines the logical start of the base area) so that logical addresses from F000 to FFFF (our window) are translated into common area 1.

For the sake of simplicity, the code stores a one-byte value. A word version would require the "index" to be multiplied by two before it is used in the computations. Use a shift to do the multiply, but beware of overflows. Multiple-dimension arrays can be programmed just as easily, but you

must compute the actual array address using the formula previously discussed.

Multiplications are slow and should be avoided; try to pick values of **Jmax** that are powers of two, and then use a series of shifts. Fast access will require some thought to optimize the computations.

One easy speed trick is shown in **NEXT1D**. Especially when gathering data, we often just put one value in the next location-random array accesses are not needed. This means we can skip all of the multiplications needed to compute the address; we just increment the last address. **NEXT1D** illustrates this approach with the one-dimensional array we've already looked at. **PUT1D** saves the current CBR and logical addresses in **SAVE\_CBR** and **SAVE\_LOG** for **NEXT1D**. After one call to **PUT1D** to set things up, all further sequential accesses can use the faster **NEXT1D**. With the single indexes shown, the speed difference is not important; with a two- or three-dimensional array, a tremendous time saving will result.

What if your program uses a number of big arrays? You can't assign more windows since the HD64180 limits mapping to three sections. Probably the easiest solution is to write a **PUT** and **GET** routine for each array, using a different **BASE\_CBR** value for each. Avoid using sequential accesses unless you can be really sure that the accesses will be restricted to one array at a time.

Large arrays are a necessary part of many software solutions. Using the memory management provided by the HD64180 can be challenging. Learning to use the MMU effectively gives you a powerful tool on a powerful processor. +

**Jack Ganssle is the president of Softaid, a vendor of microprocessor development tools. When not busy pushing electrons around, he sails up and down the East Coast on his 35-foot sloop.**

## IRS

- 2 13 Very Useful
- 2 14 Moderately Useful
- 2 15 Not Useful

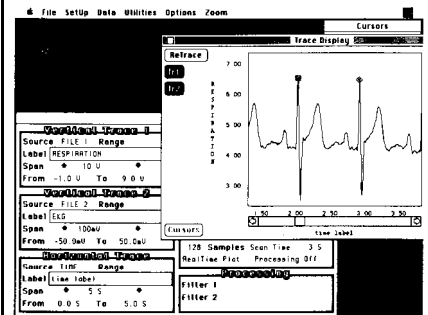
## DATA ACQUISITION SOFTWARE

We'll never leave you without a trace

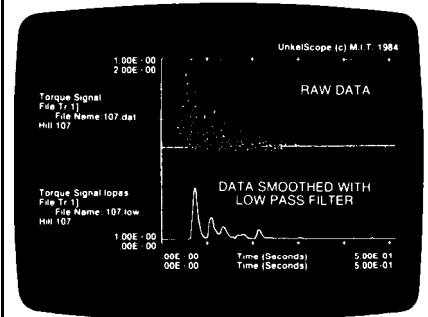
# UnkelScope®

UnkelScope is an easy-to-use, menu-driven software package that will always leave you with a clear, accurate trace. Whether you're in a laboratory or on an oil rig in the North Atlantic, UnkelScope will get the job done

- Full hardware speed
- Real-time X-Y plots
- Graphical Editing
- Data Processing
- Experiment Control
- Plus much more



MAC Version



PC Version

- UnkelScope JUNIOR \$125
- UnkelScope for MAC \$149
- UnkelScope Level 2+ \$549
- 30-day money-back guarantee

(617) 861-0181

FAX (617) 861-1850

**Unkel Software Inc.**

62 Bridge Street Lexington, MA 02173

Reader Service #178

August/September 1990 55

# FIRMWARE FURNACE

Ed Nisley

# The Furnace Firmware Project

Keypad and Piezo Beeper

Once upon a time, I built an 8031 gadget with no inputs. It would pick a witticism from EPROM and scroll the text across a 16-character LCD panel, then go to sleep for awhile. You couldn't do anything except turn it off... certainly the most (deliberately!) annoying gadget I've ever put together.

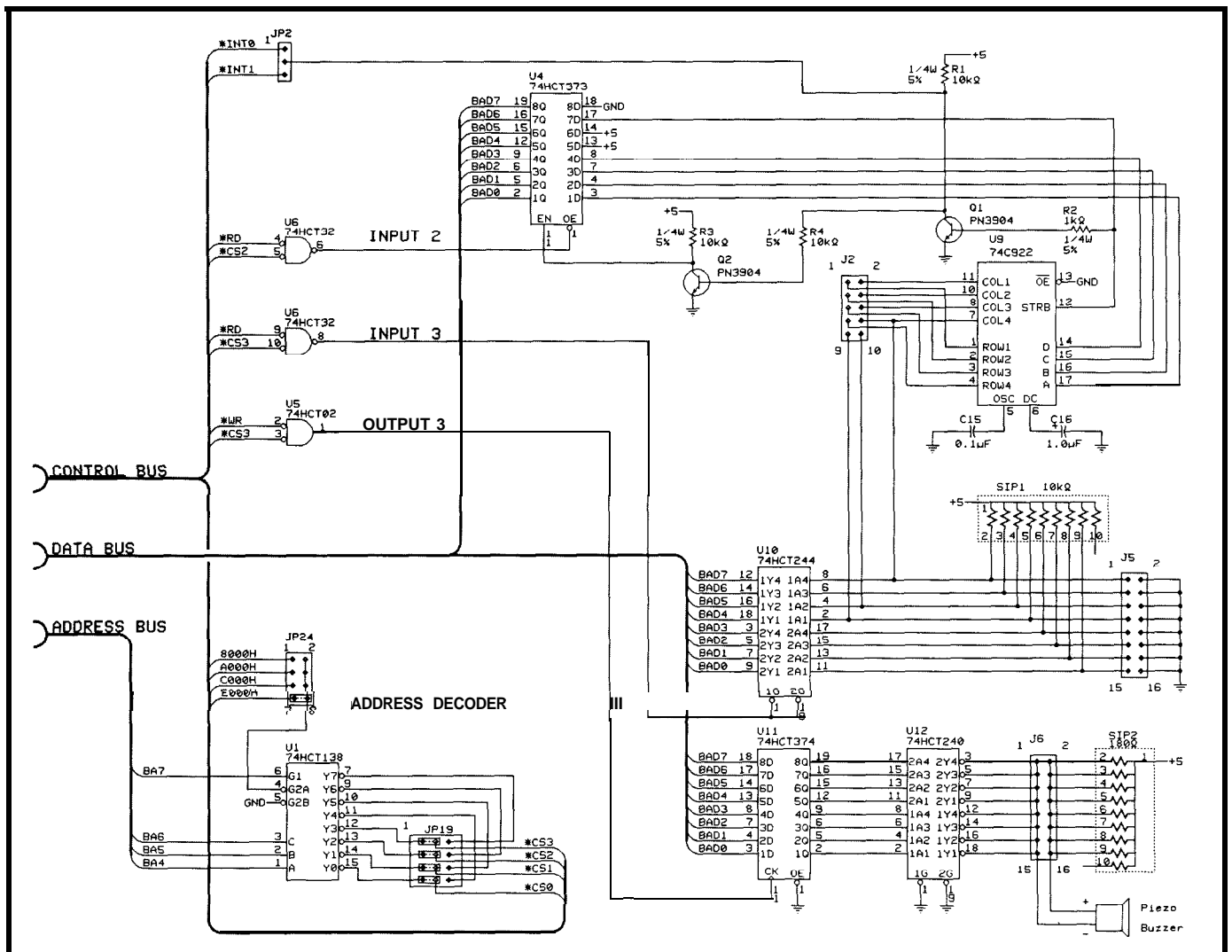


Figure 1 -The two shift keys modify the byte returned by the other 14 keys. Either or both shifts can be pressed along with any other key. Data values are shown in hex and ASCII (where prin table). You can assign any byte value to any key by changing the appropriate lookup table entry.

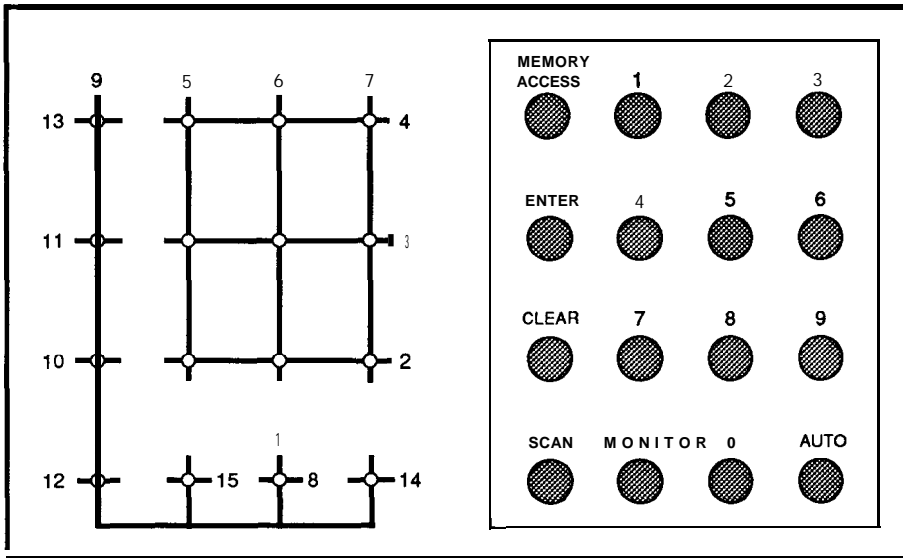


Figure 2—The 16-key keypad is laid out in a 4 x 4 matrix, but with wiring that isn't a simple 4 x 4 array.

My ongoing Furnace Firmware project, on the other hand, requires some way to set parameters, select displays, override defaults, and so forth. The LCD interface I described in the last issue uses the RTC-LCD board, so it's only natural to put the keypad hardware on that board to good use. However, in order to demonstrate some hardware and firmware tricks, I will use a keypad that requires some tinkering.

No control system is complete without an audible alarm of some sort, so this part of the project includes a piezoelectric beeper. It sounds off when you press a key, so you know when the system has accepted (or rejected!) your input.

#### HARDWARE ENCODING

The hardware starting point is the RTC-LCD keyboard interface shown in Figure 1. Under ordinary circumstances, you plug a matrix keypad into J2 and you're up and running. Because my keypad isn't quite standard, we need to understand the circuitry on the board so we know what to modify.

The 74C922 keyboard encoder scans up to 16 keys arranged in a 4 x 4 matrix. In much the same way as I described in the very first Firmware Furnace (CIRCUIT CELLAR INK, issue #1), the chip lowers one column output and examines the four row inputs

to see if any are low. If none are, it proceeds to the next column.

When you press a key, one of the row inputs goes low and the 74C922 stops scanning at that column. It puts the binary code corresponding to the key's location on the data outputs and sets the STROBE output high. Scanning resumes when you release the key, so two keys do not conflict. If you press two keys in the same column at the same time, the chip returns the code for one key or the other, but not both, and waits for you to release it before processing the other.

Jeff designed the RTC-LCD board to work with programs in either assembly language or BASIC-52, and with either polled or interrupt input. The STROBE signal latches the chip's output in the '373 register and can cause an interrupt on either INTO or INT1 depending on how jumper JP2 is configured.

The Furnace Firmware hardware needs both interrupt inputs

for other, higher-priority functions, so I am using simple polled input. Although the STROBE signal can be read through one of the INT bits, I added an extra wire to the stock RTC-LCD board to route it to data bit 6, as shown in Figure 1.

It seems as though this can't possibly work, because the STROBE signal clocks the data into the '373 latches. However, the two transistors between the 74C922 and the '373 add enough delay to ensure that the STROBE signal is latched at a logic zero when you release the key. The '373 latches track their inputs when the clock input is high, so you can read both high and low STROBE inputs right through the latch!

#### MAKING CONNECTIONS

Figure 2 shows the layout of my keypad. There are 16 keys laid out in a 4 x 4 matrix, but the wiring is not a simple 4 x 4 array. The labels printed on the keypad reflect the design of a long-gone commercial product and don't match the functions I'll need. The keypad has an adhesive backing, so mounting it to a panel is a simple matter of cutting a slot for the ribbon cable.

The numeric keys are a true 3 x 3 matrix and the "0" key can be wired

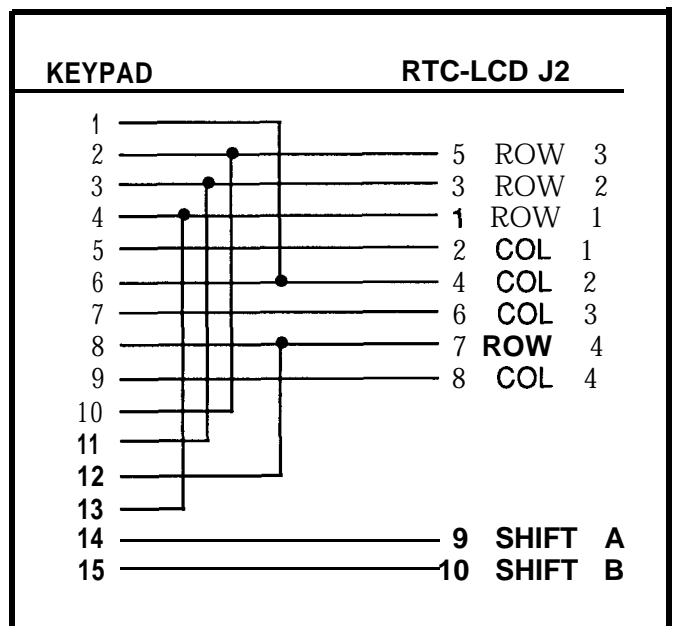


Figure 3—Attaching the keypad to the RTC-LCD board requires some cross-connection of wires.



up as the last row of the middle column. The left-hand function keys become the fourth column, but the 74C922 can encode only four of the six keys connected to the common wire. Figure 3 shows the external connections used to mate the keypad to the RTC-LCD board. Notice that the 74C922's "COL1" output connects to the second keypad column and "COL4" drives the first column of the keypad.

A quick count at J2 shows ten wires at an eight-pin (4 rows, 4 columns) connector. If you look closely at Photo 1, you'll find two additional pins embedded in a blob of hot-melt glue on the edge of the RTC-LCD board. Jeff located J2 at exactly the right spot to allow easy expansion, and it seemed a shame to miss this opportunity. Remember, I picked this keypad to illustrate some tricks; you won't have to make any changes to the RTC-LCD board to use a standard keypad.

Because the 74C922 cannot read those two added wires, the 8031 must get at them another way. The RTC-

LCD board has an X-10 interface that uses one input bit; Jeff threw in a complete 8-bit input port "just in case" someone might need it, and even included 10k pull-up resistors! Figure 1 shows the three wires leading from J2 to that port.

That port reads the column scan bit as well as the MONITOR and AUTO key "row" lines because the 8031 must know when the 74C922 is scanning that column. The key signals can only go low when a key is pushed and the column input is low. When the column signal is high, you can't tell if those keys are pressed, so the firmware must cooperate with the 74C922 to read the keypad.

The 74C922 stops scanning when it detects a keypress, so the numeric pad keys can't register until you re-

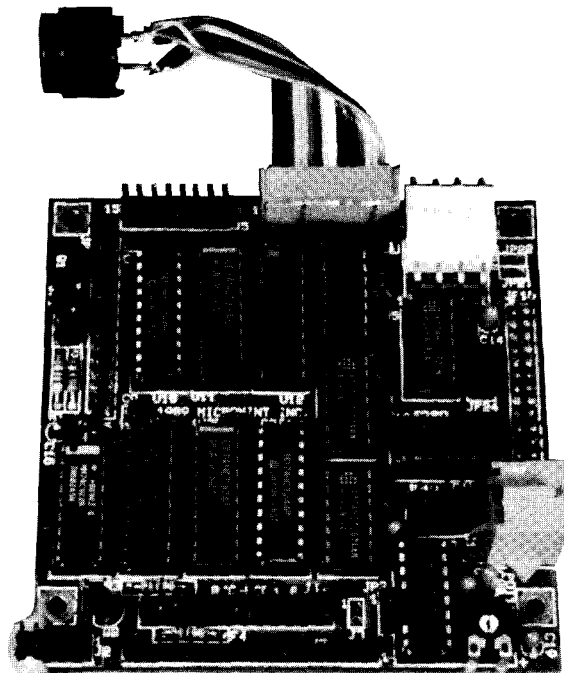


Photo 1 -A bit of extra circuitry was added to the RTC-LCD board to accommodate a nonstandard keypad.

lease them and the 74C922 resumes scanning. Because the function keys are driven by COL4, the 74C922 will scan them almost immediately after you release a key driven by any of COL1 through COL3.

The capacitor on pin 5 of the 74C922 sets the column scanning rate. The 0.1- $\mu$ F value used on the RTC-LCD board produces a 670-Hz clock, so each column scan pulse is 1.5 ms wide and the worst-case time to acquire the shift keys is about 4.5 ms after the triggering key goes up.

### SHIFTY, NOISY KEYS

KEYDEMO.BAS, shown in Listing 1, is a BASIC-52 program that reads the keypad ports and displays the bits. Lines 110 through 130 wait for a new code from the 74C922, while line 210 waits for the chip to scan the column with the two additional keys. That's all there is to it! **Editor's Note:** Software for this article is available from the Circuit Cellar BBS and on Software On Disk #16. For downloading and ordering information, see page 85.1

The whole point behind those two keys is that they can serve as shift keys. A little experimentation with KEYDEMO shows that there are 14

## Introducing a new way to think about Relaxation and Mental Proficiency...

MINDSEYE

### SYNERGIZER

Scientific studies have long associated brainwave activity with specific states of consciousness. Our normal, waking state is generally associated with Beta brainwave frequencies of 13+ hz (pulses per second). Relaxation and meditation are in the 9-12 hz Alpha range. Deep relaxation and certain kinds of mental imagery (creative thinking and "superlearning") are experienced in the 4-7 hz Theta range. Below this lies the Delta range of 1-3 hz, produced during sleep.

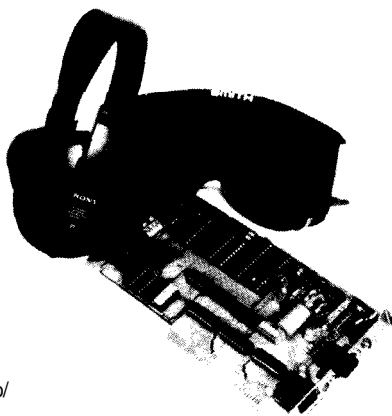
The MindsEye Synergizer is a powerful hardware/software combination that turns your IBM PCXT/AT/386 or clone into a laboratory grade audio/visual synchronizer. The Synergizer allows the user to program sessions of almost any length and complexity with each eye and ear programmed independently if desired; pulses can shift from one rate to another, while different sound frequencies are channeled left and right. Multiple time ramps and sound/light levels (over 32,000) may be included within a single programmed session. A stereo synthesizer makes available a variety of waveforms, filters and other sound parameters. The Synergizer provides more programmable capabilities than any other device available, at a remarkably low price. Requires DOS 3.0 or above; 512 K of RAM, and a hard drive are recommended.

MINDSEYE SYNERGIZER \$330 GOGGLES (required) \$65  
EXT CONTROL UNIT (optional) \$95 HEADPHONES \$35

© 1990

ORDER PHONE 800-388-6345 Credit Cards Accepted

Svnetic Svstems, Inc PO Box 95530 Seattle, Wa 98145 Ph 206-632-1722



Reader Service #173

```

50 k0=0e0a0h : x0=0e0b0h : rem keyboard and parallel I/O addr
60 k9=xby(k0) : rem set up key memory

90 print "Waiting for keypad input..." : print

100 rem - wait for keys to arrive & display what we get
110 k1=xby(k0) : rem fetch key code + strobe bit
120 if (k1=k9) goto 110 : rem wait for new key state
130 k9=k1 : rem save for next time
140 print "_____"

200 rem pick up shift keys on next scan
210 k2=xby(x0) : if 0<>(k2.and.80h) goto 210

300 rem - display the results
310 print "74C922 port is", : ph0.(k1.and.04fh)
320 print "shift bits are", : ph0.(k2.and.0b0h)

400 goto 100 : rem do it again...

```

listing 1 —KEYDEMO.BAS shows how to read the keypad, The 74C922 provides four data bits to identify one of the 14 scanned keys; the remaining two keys are read through a parallel input port.

scanned keys (0 through 9 and the whole left-hand column), each of which can have four different shift states: unshifted, either of the two shifts down at once, or both shifts pressed simultaneously. Because the 74C922 doesn't read the shift keys directly, it will not respond when you press them.

There are many ways to encode shifted keys, but there is a standard for squeezing the alphabet onto ten keys. Photo 2 should look very, very familiar, as the numeric pad is now dressed up as a push-button telephone keypad. In addition, each of the "function keys" in the left column produce four distinct codes. Those 14 keys, with a little manual dexterity, return 56 distinct codes! Figure 4 shows what code each key emits for each shift state.

KEYPOLL.BAS, shown in Listing 2, converts raw 74C922 key code and shift state data into a single byte. The trick is a translation table made from a string variable. Because the BASIC-52 manual emphasizes the "text" nature of string variables, most people don't use them for anything else, but strings can simplify many programs.

The only real difficulty is getting nonprintable ASCII data into the strings. The most readable solution uses the ASC() operator to stick a hex value into the string, a method that chews up a remarkable amount of space for a each byte. In a larger pro-

gram the setup would take a much smaller fraction of the total code and be less conspicuous.

You can load the conversion table with any values that suit your purpose. I suspect I'll change the function key values along the way, but this table will serve to get us going. Notice that it does not include such niceties as the carriage return and punctua-

tion needed to replace a standard keyboard; adding those is left as an exercise for the reader.

Incidentally, the "key pressed" beep is produced by line 210, which sends a more-or-less square wave to the X-10 parallel output port. This takes advantage of another "just in case" port Jeff included on the board. The piezoelectric beeper is tied to all eight data lines and their corresponding pull-ups, as shown in Figure 2, to get enough drive capability. Ideally, piezo beepers should be driven by a signal at their resonant frequency, usually 2 kHz, but line 210 produces about 160 Hz. As a result, the tone sounds more like a quack than a beep. Rest assured that I'll show you how to fix this.

#### BETTER KEYING

The BASIC-52 keypad code will suffice for many applications, but, as with the LCD code I presented last time, there are some distinct advantages to assembly language. Of course, the added features take more code to

# E COMMUNICATIONS

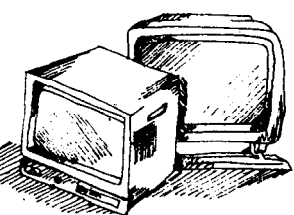


CAD MONITORS SPECIALS!

508-485-1144  
FAX 508-481-7222  
BBS 508-460-9203

## LOWBUDGETSPECIALS

<b>19" COLOR SUPER VGA MONITOR</b>	
LIKE NEW 6 MONTH WARRANTY .....	\$800.00
USED 3 MONTH WARRANTY .....	\$600.00
USED WITHOUT CASE .....	\$300 TO \$600.00
<b>19" COLOR SONY 1280 X 1024 .....</b>	<b>\$1600.00</b>
THESE SONY TRINITRONS HAVE A FULL 6 MONTH WARRANTY PARTS AND LABOR. LIMITED QUANTITY	
<b>16" IKEGAMI 64KHZ 1280X1024 NEW .....</b>	<b>\$899.00</b>
<b>16" PANISONIC 64KHZ USED .....</b>	<b>\$599.00</b>
<b>14" IKEGAMI TTL VGA CHASIS .....</b>	<b>\$249.00</b>
NEW WITH 1 YEAR WARRANTY	
<b>19" PHILLIPS 1024X800 48 KHZ GRAY SCALE .....</b>	<b>\$350.00</b>
NEW 1 YEAR WARRANTY - MAY BE ORDERED FOR VGA AT NO EXTRA CHARGE. WHEN USED IN VGA MODE THE MONITOR WILL RUN 800 X 600 X 256 GRAY SCALE OR 1024X768X16 GRAY SCALE ONLY	



CALL US ABOUT OUR LARGE VARIETY OF GRAPHIC CARDS !

194 Main ST. Marlboro, MA 01752

```

20 rem - Set up keycode conversion table
22 string 66,134 : rem the string index starts at 1, not 0!
30 $(0)="123#456#789# 0 ##AD#GJM#PTW# Q #
    #CF#ILO#SVY# Z # BE#HKN#RUX# . #!"
31 asc$(0,17)=0d1h : asc$(0,33)=0e1h
32 asc$(0,04)=081h : asc$(0,08)=082h
33 asc$(0,12)=083h : asc$(0,16)=084h
34 asc$(0,20)=091h : asc$(0,24)=092h
35 asc$(0,28)=093h : asc$(0,32)=093h
36 asc$(0,36)=0A1h : asc$(0,40)=0A2h
37 asc$(0,44)=0A3h : asc$(0,48)=0A4h
38 asc$(0,52)=0B1h : asc$(0,56)=0B2h
39 asc$(0,60)=0B3h : asc$(0,64)=0B4h
50 k0=0e0a0h : x0=0e0b0h : rem keyboard and parallel I/O addr
90 print "Waiting for keypad input..." : print

100 rem - wait for keys to arrive & display what we get
110 k1=xby(k0) : rem fetch key code + strobe bit
120 if 0=(k1.and.40h) goto 110 : rem spin if strobe off

130 rem - wait for shift scan. then combine with key code
140 k2=xby(x0) : if 0<>(k2.and.80h) goto 140
150 k2=not(k2).and.030h : rem flip shift bits
160 k1=k2.or.(k1.and.0fh) : rem combine & strip strobe

200 rem - sound a tone
210 for i=1 to 10 : xby(x0)=0 : xby(x0)=0ffh : next i

300 rem - display the result
310 print "code", : ph0,k1,
320 k2=asc$(0,k1+1)
330 print " -> char [" , chr(k2) , "]" ,
340 print " hex", : ph0,k2

400 rem - wait for key release
410 k1=xby(k0)
420 if 0<>(k1.and.40h) goto 410
500 goto 100 : rem do it again...

```

**Listing 2**—This BASIC-52 program combines the key code and shift state into a single byte. A lookup table simplifies the conversion.

accomplish, but the logic is fairly easy to understand.

Most “real” keyboards repeat a key when it’s held down for more **thana** fraction of a second. The **74C922** provides a single STROBE output when a key is first pressed, so any repetitions must be done entirely in firmware. However, we can tailor the initial delay (“holdoff”), as well as the repetition rate, to the application at hand by tweaking a few code constants.

Because the code can’t read the shift keys until the **74C922** scans their column, the only “repeatable” shifted keys are in the left column (which is wired to the shift keys). The numeric pad keyscan’t repeat because **the code** doesn’t know if the keys should be shifted before repeating them. This isn’t a problem in my application, but if all your keys must repeat in all shift states you need a different keypad. I’ll build **holdoff** and repeat into the code so you can see how it’s done, but remember the design limitations.

Another reason for assembly language is traditional: speed. While most programs don’t need blinding console I/O speed, spending too much time in “spin loops” waiting for the console to disgorge a new character is a Bad Thing because it may prevent the program from monitoring other inputs. The keypad hardware may require a few loops, such as the one that picks up the shift bits after the key is released, but using assembly code can cut those delays to the bare minimum.

Predictable timing **means** the code must use one of the on-chip hardware timers to generate regular interrupts. Because the BASIC-52 interpreter uses Timer 0 to count 5-millisecond intervals, it makes sense to measure key **holdoff** and repeat durations in multiples of 5 ms. Even if you’re not using BASIC (and the demo routines for this column don’t), it provides a convenient “standard” to follow. And, of course, timer interrupt routines had best use assembly language.

In short, while the BASIC code works, things go better with assembler.

## The DA/M<sup>TM</sup>

### The Lowest Cost Data Acquisition System

Our DA/M can solve more of your data acquisition problems at a lower price than any other product on the market. DA/M’s are used for:

- Military meteorological stations,
- Building management.
- Automated hydroponic farming,
- Industrial process control.
- Your application

In fact, DA/M’s can be used whenever you can’t afford to use any one else’s product.

Made in North America, DA/Ms are available NOW!

DATA ACQUISITION AND MANAGEMENT CORPORATION LTD.

#140, 17303 - 102 Avenue

Edmonton, Alberta, Canada T5S 1J8

Phone 1-403-486-3534

Fax 1-403-486-3535

Reader Service #1:

# INTRODUCING On-axis™

## 4 CHANNEL 24-bit QUADRATURE-MODE COUNTER

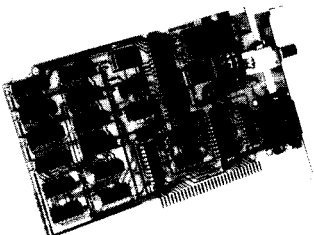
Acquires and displays position information from optical encoders. Resolution is four times the encoder. Complete with demo software and driver source code.

**\$299<sup>00</sup> INTRODUCTORY PRICE**

(add \$150 for optional connector to Bauch & Lomb "glass scales".)

# étude™

## 25 MHz 8-bit ANALOG-TO-DIGITAL CONVERTER



Based on the TRW THC1068 hybrid flash converter, its high signal-to-noise ratio yields excellent accuracy at the Nyquist limit.

- 4 KS of cache SRAM or to host as converted at DMA speed
- I/O or DMA data transfer
- 10 MHz full-power bandwidth
- 3.92 mV resolution
- Factory calibrated
- 16 jumper selectable base addresses
- External clock and trigger inputs
- TTL compatible
- Software source code included

PRICE: **\$495<sup>00</sup>**

Each product requires: PC compatible 1/2 length 8-bit expansion slot. DOS 2.11 or greater. EGA, VGA or Hercules display needed for graphic representation of data.

**Silicon Alley Inc.**

**P. O. BOX 59593  
RENTON, WA 98058  
(206) 255-7410**

©1990 Silicon Alley Inc. étude and On-axis are trademarks of Silicon Alley Inc. Other brand or product names are trademarks or registered trademarks of their respective holders. Prices & specifications subject to change.

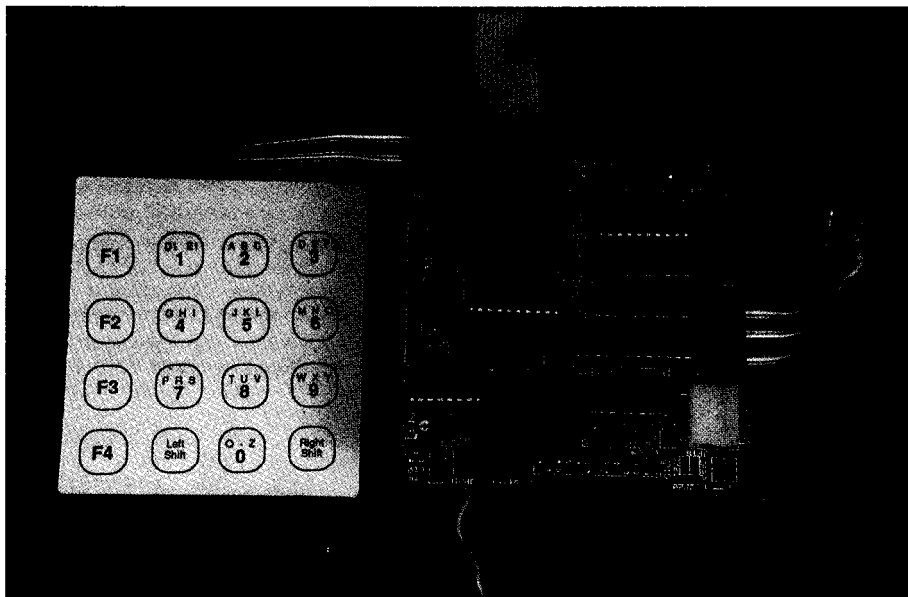


Photo 2-A combination of left and right shift keys plus function keys makes for a very powerful keyboard-like keypad.

### KEYED STATES

because the keypad interface does not use interrupts, the code must poll it to detect new keys. Most application programs cannot poll often enough to catch every key, so this task is best left to an interrupt routine. The timer ticks every 5 milliseconds, so it calls the keypad polling routine 200 times a second.

The timer interrupt handler uses 110  $\mu$ s every 5 ms, or 2.2% of the total CPU cycles. Updating the clock and handler overhead account for 42  $\mu$ s, so keypad polling requires 1.3% of the CPU horsepower. That's the tradeoff for not lashing the application program directly to the keyboard!

The keypad code cannot stall while waiting for a key release, so it must remember information between interrupts. The classic solution is a state machine that executes a specific section of code for each value of a state variable. The code changes the state variable in response to external conditions, like a new key press or a delay expiring. In general, each state continues unchanged until an event occurs; various events trigger changes to different new states.

There are three states: waiting for a new key, waiting for nonrepeating key to release, and waiting to repeat a

key. These states are represented by the value of the **CurrentState** variable. Listing 3 shows one way to implement the branches using the 8051's indirect jump instruction.

The code at **STATE 0** checks to see if there's a new key **down**. The keypad is idle most of the time, so the code bails out immediately. When a key is pressed, the code reads the shift key port to decide if the key can be repeated. If the key can't be repeated (because **COL4** is not active), the code sets **CurrentState to State 1**, which will wait for the key release and read the shift keys before converting the data.

However, if the key is repeatable, the code reads the shift key data, converts the hardware information to a character, and loads the result into the **CurrentChar** variable. It also sets the initial **holdoff** delay into a variable and selects **State 2**, which will wait for the delay to expire, load the repeat delay, and set the output variable again.

**CurrentChar** is the interface between the application program and the keypad firmware. The application reads **CurrentChar** and sets it to zero; a zero means there was no character ready. The firmware loads new characters as fast as they become available.

## BEEP!

The piezo transducer sounds best (or worst, depending on your outlook) when driven by a 2-kHz square wave. That implies only 250 microseconds between transitions, for a period of perhaps a few hundred milliseconds. But we don't want to tie up the processor in a loop during that whole time!

The solution is to reprogram Timer 0 to interrupt every 250  $\mu$ s during the beep interval and flip the output port driving the piezo beeper on every interrupt. There is no need to check the keyboard 4000 times every second, so the interrupt handler ticks a counter and calls the keyboard scanner once every 20 interrupts.

Another counter sets the total beep duration; the code ticks this counter on every interrupt and reprograms the timer to the normal 5-ms interval when it hits zero. An interface routine provides four different beep durations, one of which is the "key detected" tone. It is 30 cycles long, or about 15

SHIFT KEYS				
KEY	NONE	LEFT	L + R	RIGHT
1	311	D1	20 BLANK	E1
2	322	41 A	42 B	43 C
3	333	44 D	45 E	46 F
4	344	47 G	48 H	49 I
5	355	4A J	4B K	4C L
6	366	4D M	4E M	4F O
7	377	50 P	52 R	53 S
8	388	54 T	55 U	56 V
9	399	57 W	58 X	59 Y
0	300	51 Q	2E .	5A Z
F1	81	91	B1	A1
F2	82	92	B2	A2
F3	83	93	B3	A3
F4	84	94	B4	A4

Figure 4—The 14 keys on the keypad add up to produce 56 distinct keycodes, enough to implement a full-featured keyboard.

ms, and, believe me, this is one real beep!

### RELEASE NOTES

The KBDTEST.HEX file scans the keypad and sends characters to the

serial port and to the LCD driver routines described last issue. Remember, the left column returns values over 80 hex, so the LCD and CRT displays are different. If you don't have an LCD, the bits will drop off harmlessly and you'll have to watch your CRT.

## We got you covered...

... with complete embedded system support for popular PC compilers on Intel 80x 86 and NEC V-Series microprocessors. Now you can develop an embedded application with your choice of compiler—with full support for source level debuggers and in-circuit emulators.

Paradigm LOCATE supports popular C, Pascal, BASIC & Modula-2 compilers from Microsoft, Borland and others.

- comprehensive user's manual
- compiler startup code & examples
- free technical support
- math coprocessor emulation support
- extensible MS-DOS emulator
- full Intel OMF output

Lack of an emulator got you down? Not a problem with our Turbo Debugger interface option. Now you can debug your target hardware from the comfort of your PC.

**Paradigm LOCATE \$295.00**  
**Turbo Debugger Interface \$195.00**

Satisfaction Guaranteed  
 Orders: (800) 537-5043  
 Technical Support: (508) 478-0499  
 BIX: join paradigm  
 Visa/Mastercard/C.O.D. Accepted

**Paradigm Systems**  
 P.O. Box 152 Milford, Massachusetts 01757  
 Turbo Debugger is a registered trademark of Borland International



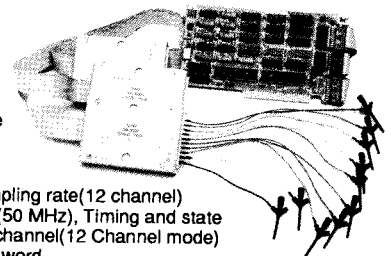
Reader Service #163

## POWERFUL • AFFORDABLE INSTRUMENTS

### 100 MHz Logic Analyzer

**\$799**  
**LA12100**

Price is complete Pods and Software included

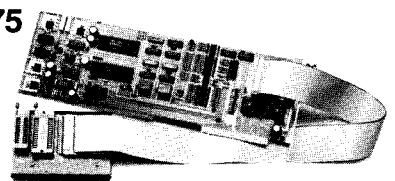


- 100MHz sampling rate(12 channel)
- 24 Channels(50 MHz), Timing and state
- 2K samples/channel(12 Channel mode)
- 24 Bit trigger word
- TTL threshold level
- Internal and External Clocks
- Menu driven software
- FREE software updates on BBS

• More sophisticated units also available

### almost UNIVERSAL PROGRAMMER

**PAL \$475**  
**GAL**  
**EPROM**  
**EEPROM**  
**PROM**  
**87xxx...**



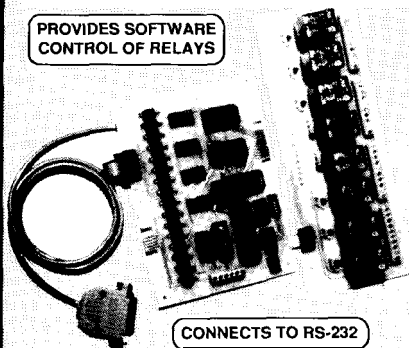
- 20 and 24 pin PALS, EPLDs
- 16V8, 20V8, 22V10 GALs
- 2716-27020 EPROMs
- 87xxx MICROS
- EEPROMs(incl. 8 pin serial)
- Byte Split/Merge(16 & 32 bit)
- JEDEC, INTEL HEX, Motorola 'S' files
- Dallas NVS RAM programming
- PC/XT/AT COMPATIBLE
- FREE software updates on BBS

Call - (201) 994-6669  
**Link Computer Graphics, Inc.**  
 4 Sparrow Dr., Livingston, NJ 07039 FAX:994-0730

Reader Service #145

# RELAY INTERFACE

PROVIDES SOFTWARE CONTROL OF RELAYS

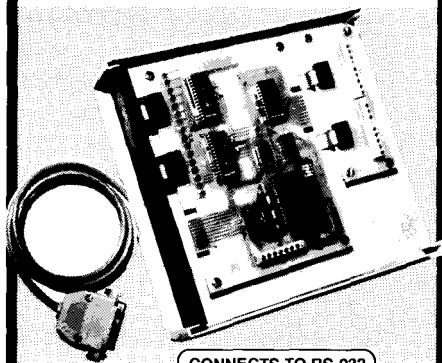


CONNECTS TO RS-232

## AR-16 RELAY INTERFACE \$ 89.95

Two 8 channel relay output ports are provided for control of up to 16 relays (expandable to 126 relays using EX-16 expansion cards). Each relay output port connects to a relay card or terminal block. A variety of relays cards and relays are stocked. call for more info  
 FID-6 REED RELAY CARD (8 relays) ..... \$ 49.95  
 FID-6 RELAY CARD (10 amp SPDT 277 VAC) \$69.95  
 EX-16 EXPANSION CARD (16 channel) ..... \$59.95

# ANALOG TO DIGITAL



CONNECTS TO RS-232

## ADC-16 (16 channel) \$99.95

Input temperature, voltage, amperage, pressure, energy usage, energy demand, light levels, joystick movement and a wide variety of other types of analog signals. Inputs may be expanded to 128 status inputs using the ST-32 expansion cards and up to 112 relays may be controlled using EX-16 expansion cards. Analog inputs may be configured for temperature input using the TE-6 temperature input card. (enclosure, terminal block and cable sold separately)  
 SJ-32 STATUS EXPANSION CARD \$ 79.95  
 Input on/off status of switches, thermostats, relays, security devices, smoke detectors, HVAC equipment and hundreds of other devices. The ST-32 provides 32 status inputs (opto Isolators sold separately).  
 TE-6 TEMPERATURE INPUT CARD \$ 49.95  
 Includes 6 solid state temperature sensors and 8 calibration trimmers. Temperature range is minus 76 to 145 degrees F. Very accurate.

FULL TECHNICAL SUPPORT...provided over the telephone by our staff. A detailed technical reference manual is provided with each order including programming examples on disk in Basic, C and Assembly Language.

HIGH RELIABILITY...engineered for continuous 24 hour industrial applications.

Use with IBM and compatibles, Tandy, Apple and most other computers with RS-232 or RS-422 ports. All standard baud rates and protocols may be used (50 to 19,200 baud) default is 9,600 baud 6 data bits, 2 stop bits, no parity.

Use our 600 number to order free Information packet. Technical Information (614) 464.4470

24 HOUR ORDER LINE (800) 642-7714  
 Visa-Mastercard-American Express-COD

ELECTRONIC ENERGY CONTROL, INC.  
 360 South Fifth Street, Suite 604  
 Columbus, Ohio 43215

Reader Service # 114

```
;- keyboard scanner state branch table
; these are real jumps, not just addresses!
StateTable EQU $
LJMP State0
LJMP State1
LJMP state2
NUMSTATES EQU ($-StateTable)/3 ; # of state entries

KbdScanner PROC
;- branch to state machine routines
GetXData CurrentState ; fetch state variable
CJNE A, #NUMSTATES, L?1 ; C set if A < NUMSTATES
L?1 JNC State0 ; flush on error

MOV B, #3 ; turn into table index
MUL AB
MOV DPTR, #StateTable ; point to table base
JMP @A+DPTR ; and dive into the

routine

;- State 0: key wasn't pressed, so check for new one
; if shift scan is active, we start holdoff & repeat
State0 GetCWord DataAddr ; fetch code + strobe
MOVX A, @DPTR
PutXData KeyCode ; save for later
ANL A, #STROBEMASK ; isolate strobe bit
XRL A, #STROBEFLIP ; and make 1 = active
JNZ L?newkey ; if zero, still no key

CLR A ; no key, flush char
PutXData CurrentChar
PutXData CurrentState ; in case of state error
JMP L?done

L?newkey GetCWord ShiftAddr ; holdoff/repeat?
MOVX A, @DPTR ; (get shift scan state)
JB ACC.SHIFTSCANBIT, L?norep ; high=not scanned

MOV B, A ; save shift state
GetXData KeyCode ; fetch code again
CALL KbdTranslate ; make it a character
PutXData CurrentChar ; set it in the output
PutXData StoredChar

GetCData HoldOff ; set up initial holdoff
PutXData DelayCounter

MOV A, #2 ; go to state 2,
PutXData CurrentState ; wait for repeat
JMP L?done

L?norep MOV A, #1 ; go to state 1,
PutXData CurrentState ; wait for release
JMP L?done

<<< Code for states 1 and 2 is omitted >>>
```

listing 3—The timer interrupt handler calls the keyboard scanner routine every 5 ms. The keyboard code uses a state machine to remember information between calls.

The BBS files for this column include revised LCD code along with new timer and keyboard routines. Make sure you don't mix the versions, since the old code doesn't handle all the new entry points. The EPROM area holding the constants now includes keyboard and timer setup values.

Next time around, I'll look at measuring temperatures, keeping the time of day, and storing data in non-volatile RAM. ❖

Ed Nisley is a member of the Circuit Cellar INK engineering staff and enjoys making gizmos do strange and wondrous things. He is, by turns, a beekeeper, bicyclist, Registered Professional Engineer, and amateur raconteur.

The RTC-LCD schematic is printed by permission of Micromint Inc.

## IRS

216 Very Useful  
 217 Moderately Useful  
 218 Not Useful

# Creating a Nonvolatile RAM Module

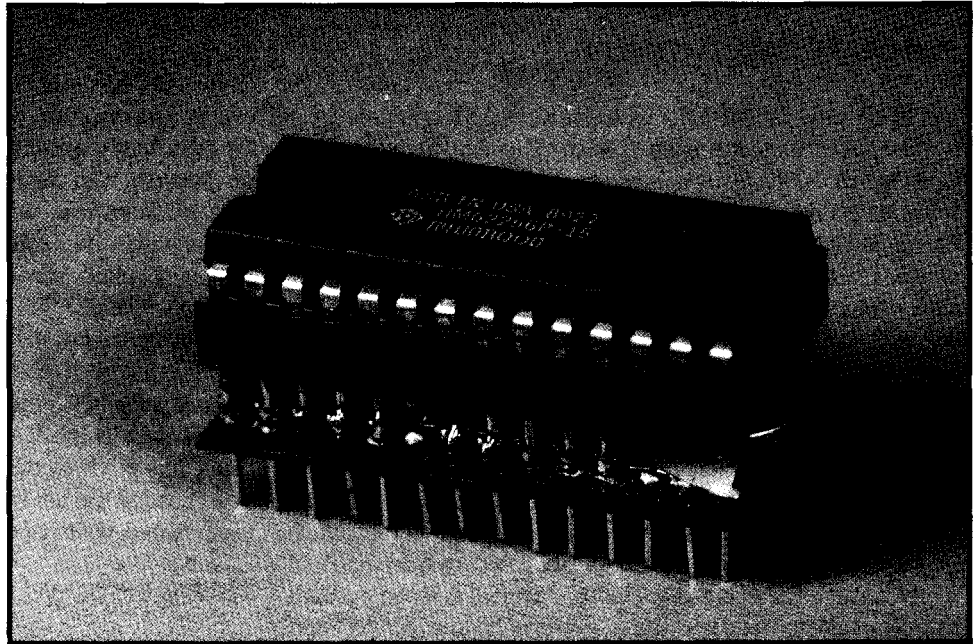
FROM  
THE  
BENCH

Jeff Bachlochi

I'm sure you've had your thought process interrupted many times. It's an everyday occurrence and shouldn't destroy your concentration. Take this morning for instance: I was working on a problem when Mary's voice clashed with the tune I was humming. "Did you get a chance to look at that new ZIF socket?" she queried. "Yes, I did and it's too hard to open, unless you've long fingernails." We discussed the problem further and I was ready to resume my quest.

"Now where was I going?" I puzzled. For what seemed like an eternity, I stood there, my mind doing a playback of the events leading up to the present. I could visualize a journey along the neural highway within my head, speeding from one neuron to the next, across each appropriate synapse. "Let's see... a new project... prototype... parts installation... not finished... why? .. SOLDER, yeah that's it. I ran out of solder!" My mind was partying over the accomplishment. It seems as though even the chemical memory of our minds can lose a bit from time to time.

Everyone has seen the message "MEMORY ERROR" pop up on the computer's video screen. This is usually followed by glazed eyes, an immovable stare, and a jaw



which falls clear to the floor. Eventually, a push of the reset button allows one to rewind and start over, but the consequences are usually the permanent loss of some data. You have to wonder, was this due to a device failure or a program bug. Hardware types will probably feel at ease blaming the problem on a program bug, while software gurus know it's a design flaw.

I consider memory the most important part of a system. It continues to shrink in size and cost. I wish I had a buck for every time I've heard, "(fill in your favorite number here)K: I'll never need more than that!" I remember when that number was in bytes, now it's kilobytes and even mega/gigabytes.

Most PCs use dynamic RAM for temporary storage, but the faster machines are using static RAM for speeds necessary in caching. I happen to prefer static RAM even though it requires more room per bit than dynamic RAM. It's easier to use and requires less power. Unfortunately, even though its name-static-suggests it can retain data without power, that just ain't so. Once power is removed, static RAM forgets everything you taught it.

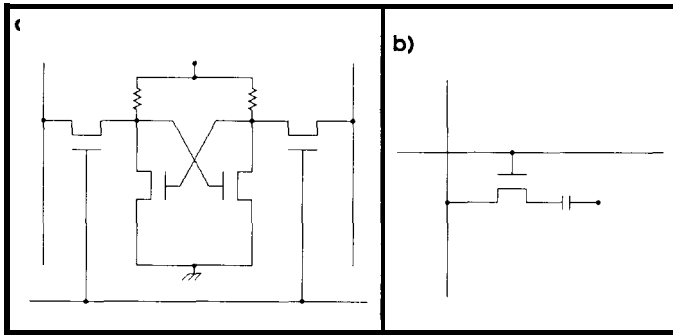


Figure 1 —(a) Static RAM cells are made up of transistors organized in a flip-flop arrangement. (b) Dynamic RAM cells rely on charge stored on a capacitor to retain data, so must be periodically refreshed.

## TEMPORARY DATA RETENTION

Each cell or bit within a static RAM consists of a number of transistors organized in a flip-flop arrangement (see Figure 1a). Logic levels are set and held by the flip-flop without having to recharge or refresh the cell, as in dynamic RAMs (see Figure 1b). This feature requires more physical area **per bit** (a four-transistor static cell has roughly four times the area of a one-transistor dynamic cell). You can begin to see why static RAMs are so much larger per bit than dynamic RAMs.

Operating current for most static RAM is in the milliampere range. A 100-mAH battery could sustain a static RAM at 10 mA for 10 hours—hardly worth considering. However, static RAM does have a data retention mode in which the current requirements are reduced to **microamperes**. This varies greatly from 100  $\mu\text{A}$  down to 1  $\mu\text{A}$  in some cases. (Check data sheets carefully since not all manufacturers produce these extremely low power versions.) At a data retention level of 1  $\mu\text{A}$ , the 100-mAH battery will sustain life for  $100/0.001$  or 100,000 hours. That's over 4000 days or in excess of 11 years. Now we are talk'n possibilities!

## EXTENDED DATA RETENTION

EPROMs are an accepted (semipermanent) form of data retention. Semipermanent not because they can be erased, but because **they** rely on a memory cell which holds a charge that eventually leaks off, losing the data. I don't mean to frighten anyone with this but someday your computer will forget its own name—sort of like solid-state Alzheimer's disease. By that time, though, it will be collecting dust like my TRS-80 Model I, so it really won't matter.

Actually, at room temperature, a covered EPROM will retain its charge for thousands of years. Left uncovered under indoor fluorescent lights, it would take only three years to lose its charge. Outdoors and exposed to the sun, an EPROM could be erased in as little as a week. It is no wonder that high-intensity UV erasers can pound those electrons free in as little as ten minutes.

EEPROMs can give you a read/write version of an EPROM using only 5V. They don't require the higher

programming voltages of EPROMs but are very expensive and do require special write timings. This means they are not a drop-in replacement for the read/write abilities of today's static RAM.

This leads us back to the static RAM for data retention. Why all this talk of data retention? Many projects require the collection and storage of data even after the power goes down. The loss of power may be deliberate, to save energy when using batteries, or unexpected from an interruption of line voltage. When the system returns to operation, your data must be fully intact.

Program development is another area where data retention can come in handy. Most microcontrollers start executing code upon power-up (the 8031 starts executing code at 0000H). Many micros with a built-in high-level language come up in a command mode, ready for user input (the 80C52 executes BASIC). If the processor finds the necessary information stored at a particular address, it will automatically enter the execution mode and run the program written in the high-level language. This auto-execution information, whether it be machine language or a high-level code, is generally stored within an EPROM. A battery-backed static RAM can serve as a replacement device for the EPROM. This lets the user skip all the steps involving Intel hex files, EPROM programmers, and ultraviolet erasers. If using a read/write device as read-only memory makes you cringe a bit, you can add a **write-protect** tab to the RAM. No, it's not a joke, although it is more like 8" diskette protection. Remember those, where you remove the tab (in our case a jumper) to protect the device from a write operation?

## POWER-UP/DOWN PROTECTION

There are two main problems in protecting the contents of static RAM during a power fluctuation. First, whenever normal power is interrupted,  $V_{cc}$  of at least 2 volts must be maintained on the static RAM. Second, write protection must be maintained during times of insufficient  $V_{cc}$ . Normal  $V_{cc}$  would be 4.5-5.5 volts, which is 5 volts  $\pm 10\%$ ; or 4.75-5.25 volts, which is 5 volts  $\pm 5\%$ . Detecting out-of-tolerance  $V_{cc}$ , protecting an unauthorized write attempt, and switching over to battery back-up can be accomplished by one device: the Dallas DS1210.

The DS1210 requires only 100 mA for operation. See Figure 2 for a **pinout**. Pin 1 is  $V_{cc}$  output to the static RAM. It will switch between system power and battery voltage for data retention when system power is out of tolerance.

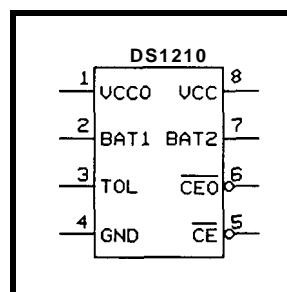


Figure 2—The Dallas Semiconductor DS 12 10 handles all the details of switching from the regular power supply to backup batteries in the event of a power loss.



Pins 2 and 7 are inputs for external batteries. The chip will use whichever battery is of a higher voltage, allowing for replacement of the batteries one at a time without loss of memory. Only one battery is necessary for operation. The tolerance of Vcc can be selected by tying pin 3 (TOL) high for less than 4.5-volt detection or low for less than 4.75-volt detection. The DS1210 uses pin 4 for ground. Pin 5 is an input for the static RAM's normal chip select. Pin 6 is the protected chip select output to the static RAM.

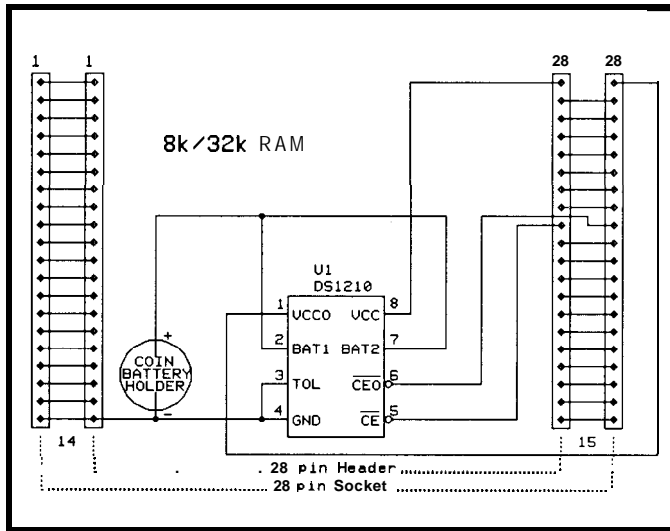


Figure 3—The DS1210 makes constructing a battery backed up socket for RAM chips easy.

The rerouting of the RAM's normal chip select signal through the DS1210 allows illegal chip selects to be discarded, thereby protecting the data held within the static RAM. Pin 8 provides system power to the DS1210. It is this pin which will be monitored for an out-of-tolerance condition.

In addition to these functions, the DS1210 can alert the system to a low-battery condition. This is accomplished by doing a nondestructive test on any memory address within the static RAM immediately after a power-up. Read a

location and save this value for replacement later. Write some other value (the value's complement is a good choice) and try to read it back. If the write/read compares, then the battery is OK. If not, then the DS1210 will block chip select during the read operation and the value read back will not match. The read control is only affected once-on the second read after power-up.

See Figure 3 for a quickie nonvolatile controller. I chose to make a few additions which allow the

module to work with 8K and 32K RAMs and let the RAM module pinout directly resemble an 8K or 32K EPROM. This allows the RAM module to be loaded with code from a read/write space and moved to a ROM socket totally protected from accidental writes (see Figures 4 and 5).

#### DANGER-CONSTRUCTION AREA

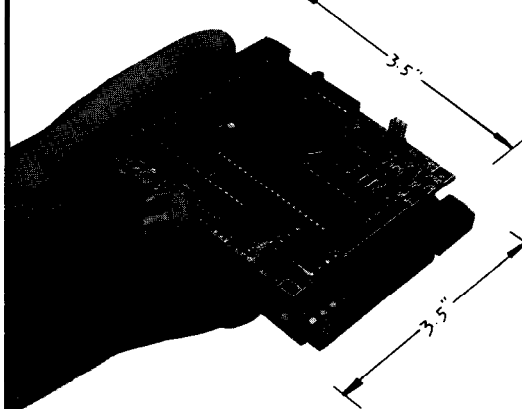
If you are planning to build the simple nonvolatile RAM module, you will need the parts shown in Table 1.

## MICROMINT Introduces "Micro" Controlling!

After years of experience in manufacturing OEM controller boards and talking to customers, we think we have hit upon just the right combination of format and function to satisfy even the toughest case of "relay mentality." Realizing that not every computer/controller application warrants a Cray XMP, Micromint offers a tiny 8031/8052-based controller board for those dedicated and cost-sensitive installations.

New MC-NET™ software links your desktop up to 31 RTC controllers.

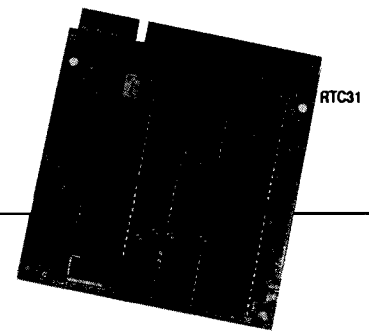
STACKED  
RTC31 & RTCIO



#### RTC31 and RTC52 Technical Specifications

- 8031 processor (RTC31) or Micromint 80C52-BASIC processor (RTC52)
- 11.05-MHz system clock
- Uses 8K or 32K memory chips
- Up to 64K bytes of RAM or EPROM
- 5-volt-only operation
- 110-19200 bps RS-232 and/or RS-485 serial port
- Use stand-alone or networked
- 12 bits of parallel I/O
- Vertical-stacking expansion bus
- Screw terminals or quick disconnects
- Small 3.5"x3.5" format
- 30 mA typical operating current (RTC52)

RTC31-1 8031 Controller	\$119.00
RTC31-1 OEM 100-Quantity Price	\$79.00
RTC52-1 80C52 Controller	\$139.00
RTC52-1 OEM 100-Quantity Price	\$99.00



#### RTCIO Technical Specifications

- Three bidirectional parallel ports (24 bits)
- 8-channel, 8-bit ADC (D-W); 9,000 samples/sec (optional 4-channel, 10-bit ADC)
- 4-channel, 8-bit DAC (D-W); 2-μs response time
- Battery-backed clock/calendar and presetable time-interrupted capability
- DC to DC conversion—5-volt-only operation
- Screw terminals or quick disconnects
- Small 3.5"x3.5" format

RTCIO RTCIO board with parallel I/O and A/D converter	\$119.00
RTCIO OEM 100-Quantity Price	\$89.00

#### Also Available

RTC-OPT0 B-channel Optoisolated I/O Expansion Board	single qty. \$139.00
RTC-SIR Serial, Timer, and Infrared I/O Expansion Board	single qty. \$149.00
RTC-LCD LCD, Keyboard, and X-10 I/O Expansion Board	single qty. \$99.00
RTC180-1HD64180 Controller Board. 96K memory; 1024 bits EEPROM; 24 bits TTL I/O; 8-channel, 10-bit ADC; 2 serial ports	starting at \$219.00



MICROMINT, INC.

4 Park Street, Vernon, CT 06066

Te2 (203) 871-61708. Fax2 - 2 2 0 4

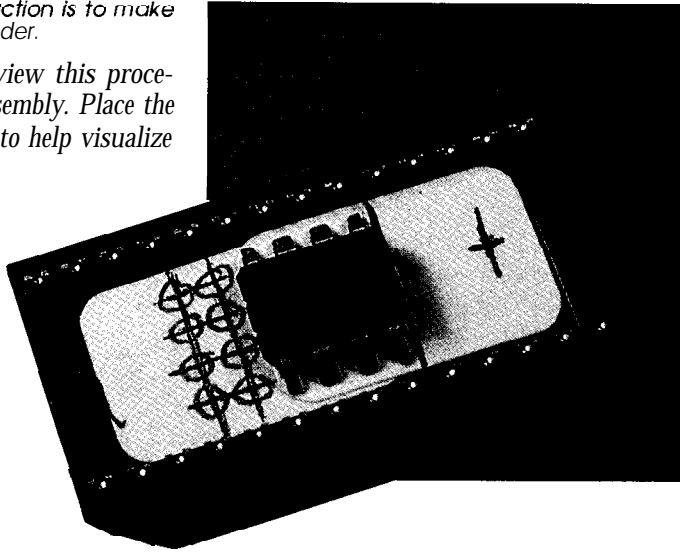
Reader Service #156

**Photo 1**—The first step in construction is to make sure everything will fit on the header.

*[Author's Note: Please review this procedure completely prior to any assembly. Place the various parts upon one another to help visualize how things should fit.]*

Construction begins with **28-pin** header/plug. Stick a piece of masking tape between the pins on the header and mark the eight spots where the square pin header will be mounted into the pin-1 end of the header. The pins should not protrude past the end of the header.

Since the coin battery holder is too large to fit into the header, remove the contacts from the holder. Mark the mounting positions for the battery contacts on the masking tape at the opposite end of the header. Check to verify the DS1210 will fit between the right-angle pin-header and the coin battery contact before putting any holes into the **28-pin** header. See Photo 1.

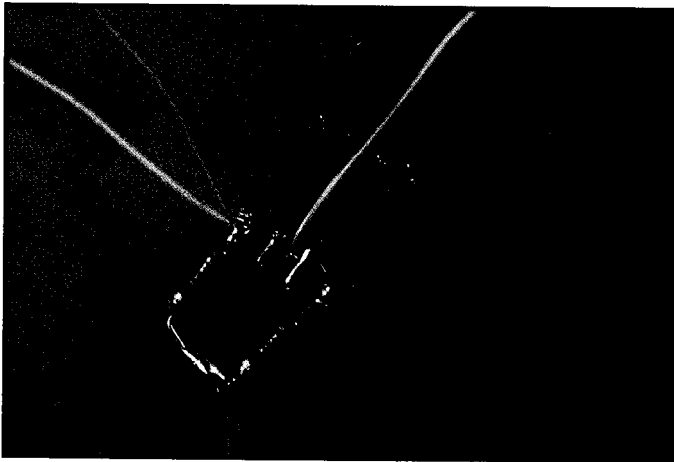


The holes can be either drilled, if you have access to small bits, or melted by using a paper clip or other appropriately sized piece of metal. I used a piece of wire wound around my soldering iron's tip.

Prepare the right-angle configuration header using wire-wrap wire and quarter-watt resistors. See Photo 2. Superglue the right-angle pins into the **28-pin** header. Next, solder a piece of wire-wrap wire to each of the coin battery contacts. Insert the (-) contact of the bat-

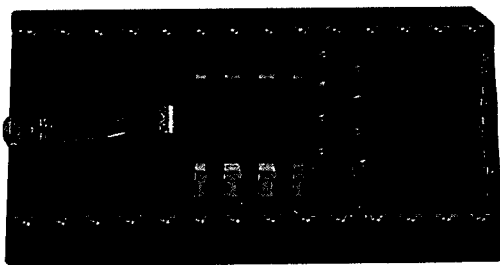
tery holder (the **center one** that looks like a nail) into the **28-pin** header and bend over the stem on the bottom of the header. Put an extra **90-degree** bend in the (+) contact (the long contact which holds the battery against the (-) contact) about **1/8** inch below the present corner forming a "C." This part is made of hardened steel to hold its springiness, so don't ruin your favorite cutters when trimming the excess leads on the bottom of the header. Trim the leads to prevent the battery contacts from shorting to one another.

Now trim the ends off each pin of the DS1210 so that it will lay flat on the header. Superglue the **DS1210** be-

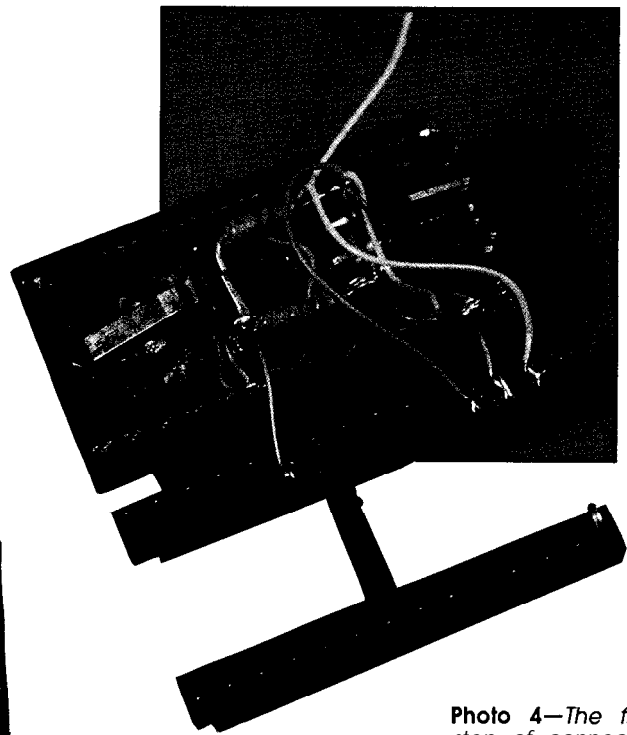


**Photo 2**—All the socket configuration is accomplished through four 2-pin jumper headers.

(If you choose to wire the header without the configuration jumpers, you may wish to add a second coin battery for double the backup capacity. Wire each battery to its own input pin on the **DS1210**.)



**Photo 3**—Only the spring clip from the battery holder is used with the final socket.



**Photo 4**—The final step of connecting everything together is by far the trickiest.

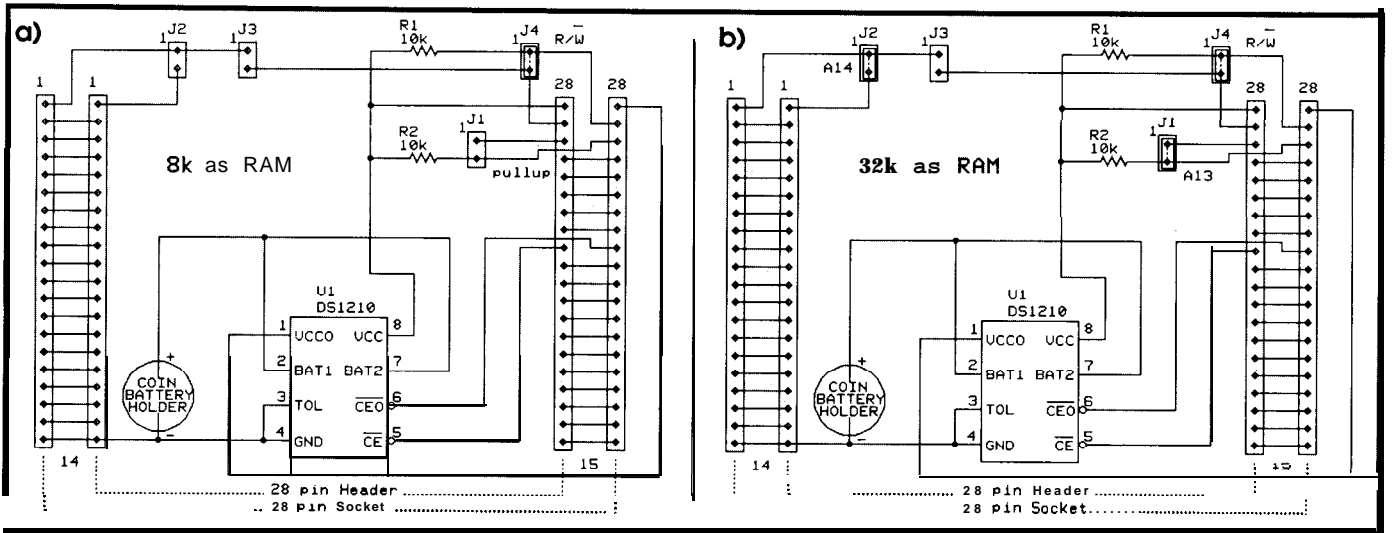


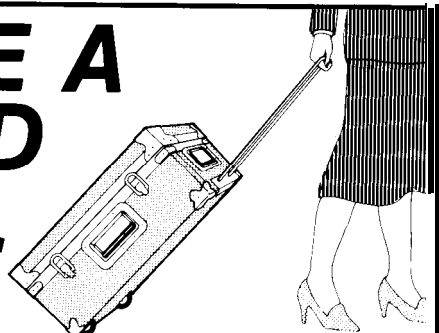
Figure 4—A pair of jumpers on the basic battery backed up socket allow the use of either 8K (a) or 32K (b) RAM devices on the same socket.

tween the right-angle header and coin battery contacts. Use a bit of glue to help hold the battery contacts in place. See Photo 3.

If you refer to the schematic, you will see that most of the IC socket's leads will connect directly to the 28-pin header. However, five out of the 28 pins will not make direct connections. These five connections are interrupted by the configuration jumpers and battery back-up control. First solder the ground from the (-) battery contact to pin

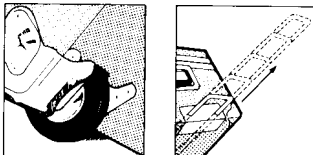
14 of the 28-pin header and pins 3 and 4 of the DS1210. Next solder the wire from the (+) battery contact to pins 2 and 7 of the DS1210. Now solder one lead from the pull-up resistors to pin 8 of the DS1210 and the other lead to pin 28 of the 28-pin header. Solder a wire from pin 20 of the 28-pin header to pin 5 of the DS1210—this is the CE input. Solder the appropriate wires to pins 1, 26, and 27 of the 28-pin header all coming from the right-angle configuration header.

# TAKE A LOAD OFF...



A rugged CABBAGE CASE? lined with plenty of foam for your equipment can **TAKE A LOAD OFF YOUR MIND** when you've got to travel.

**TAKE A LOAD OFF YOUR BACK** with our exclusive tilt-wheels and extension handle option.



**UNLOAD ON US!**

Call or write to tell us about your shipping or carrying problems **WE HAVE SOLUTIONS!**



CABBAGE CASES, INC.  
1166-C STEELWOOD ROAD  
COLUMBUS, OHIO 43212-1356  
(614) 486-2495 FAX (614) 486-2788  
(800) 888-2495

Reader Service #111

# Total control with LMI FORTH™

Programming Professionals:  
an expanding family of compatible, high-performance compilers for microcomputers

#### For Development:

Interactive Forth-83 Interpreter/Compilers for MS-DOS, OS/2, and the 80386

- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 500 page manual written in plain English
- Support for graphics, floating point, native code generation

#### For Application: Forth-83 Metacompiler

- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, 6303, 6809, 68HC11, 34010, V25, RTX-2000
- No license fee or royalty for compiled applications



Laboratory Microsystems Incorporated  
Post Office Box 10430, Marina del Rey, CA 90295  
Phone Credit Card Orders to: (213) 306-7412  
FAX: (213) 301-0761

Reader Service #144

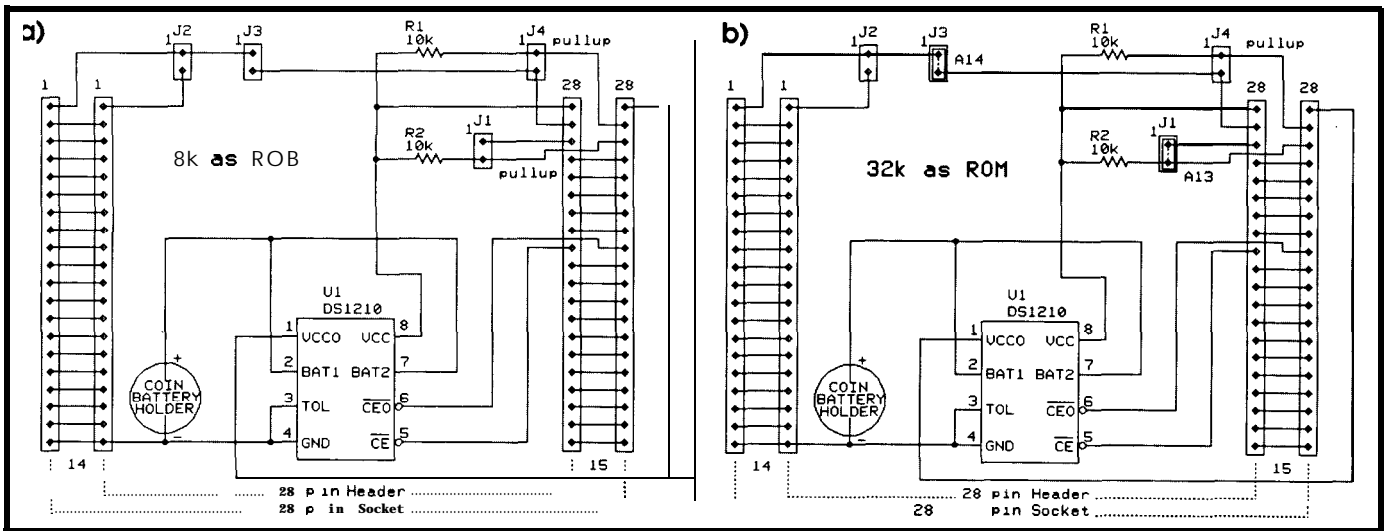


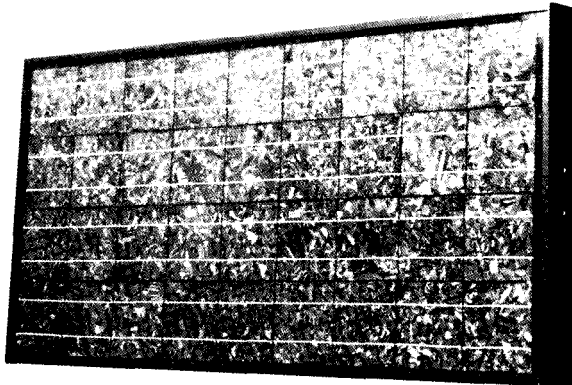
Figure 5-A second pair of jumpers allows the developer to write-protect the RAM in the socket allowing it to look like a ROM. Again, both 8K (a) and 32K (b) parts are supported.

Dress the wiring away from any pins to prevent shorts which may be caused by the melting of insulation when the pins are heated. There will not be much room between the IC socket and the 28-pin header, so keeping the wiring short will help to ease assembly.

Remove both of the end cross pieces from the IC socket so that the two sides are held together only by the center cross member. Bend the five pins (1, 20, 26, 27, and 28) inward so that they will not make contact with the 28-pin

header when the two are mated together. Lay it upside-down and next to the 28-pin header, so that pins 15-28 of each device are next one another. See Photo 4. Solder a short wire from pin 6 of the DS1210 to pin 20 of the IC socket-this is the CE output control. Solder the two appropriate wires from the right-angle header to pins 26 and 27 of the IC socket. Solder a short piece of wire from pin 1 of the DS1210 to pin 28 of the IC socket-this is the switched Vcc source which will power the RAM. One wire

## ELECTRICITY from SUNLIGHT



Photovoltaics for remote and online power applications. Reliable, renewable electricity from sunlight. Where there is a battery to be charged, there is a place for photovoltaics.

## SUNNYSIDE SOLAR

RD4 Box 808 Green River Road  
Brattleboro, Vermont 05301

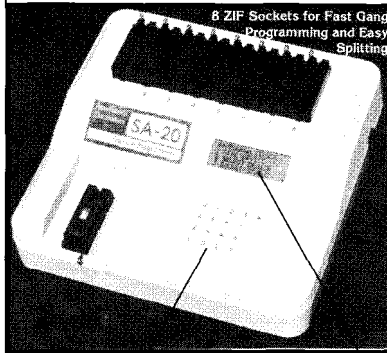
802-257-1482

or circle 172 on the Reader Service Card

## EPROM PROGRAMMERS

Stand-Alone Gang Programmer

\$750.00



8 ZIF Sockets for Fast Gang Programming and Easy Splitting

- Completely stand-alone or PC driven
- Programs E(EP)ROMs
- 1 Megabit of DRAM
- User upgradable to 32 Megabit
- .3/6" ZIF socket, RS-232, Parallel In and Out
- 32K internal Flash EEPROM for easy firmware upgrades
- Quick Pulse Algorithm (27256 in 5 sec, 1 Megabit in 17 sec.)
- 2 year warranty
- Made in USA
- Technical support by phone
- Complete manual and schematic
- Single-Sock.91 Programmer also available. \$550.00
- Split and Shuffle 16 & 32 bit
- 100 User-Definable Macros, 10 User-Definable Configurations
- Intelligent Identifier
- Binary, Intel Hex, and Motorola S

20 Key Tactile Keypad (not membrane)

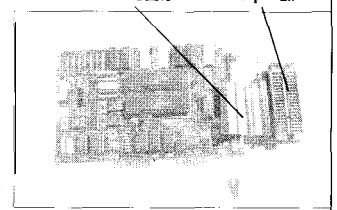
20 x 4 Line LCD Display

Internal Programmer for PC

\$139.95

New Intelligent Averaging Algorithm Programs 64A in 10 sec., 256 in 1 min., 1 Meg (27010, 011) in 2 min 45 sec., 2 Meg (27C2001) in 5 min. Internal card with external 40 pin ZIF

- Reads, verifies, and programs 2716, 32, 32A, 64, 64A, 128, 128A, 256 512, 513, 010, 011, 301, 27C2001, MCM 68764, 2532
- Automatically sets programming voltage
- Load and save buffer to disk
- Binary, Intel Hex, and Motorola S formats
- Upgradable to 32 Meg EPROMs
- No personality modules required
- 1 year warranty • 10 day money back guarantee
- Adapters available for 8748, 49, 51, 751, 52, 55, TMS 7742, 27210, 57C1024, and memory cards
- Made in USA



NEEDHAM'S ELECTRONICS

Call for more information

4539 Orange Grove Ave - Sacramento, CA 95841  
Mon Fri 8am - 5pm PST

(916) 924-8037  
(916) 972-9960

C.O.D. FAX VISA

Quan	Part#	Vendor	Description
1	BH500-ND	Digi-Key	12mm Coin Cell Holder
1(2)	P183	Digi-Key	Coin Battery (38 mA <sup>H</sup> )
1(2)	28HP	Jameco	28-pin Header Plug
1	DS1210	Dallas Semi	Power Monitor
1	24-pin IC Socket	(open-body construction)	
<i>Add these items for the universal version RAM/ROM module:</i>			
1	923872R	Jameco	2x4 R-Angle Sqr-Pin Header (available in snap-off strips)
4	JOPEN	Jameco	Shorting jumpers
4	R10K	Jameco	10k 1/4W 5% Resistors

Table I-All of the parts necessary to make your own battery-backed RAM socket are readily available.

remains from the right-angle configuration header. To keep the leads short (remember, all this has to be crammed between the sockets), rotate the IC socket atop the 28-pin header before soldering it to pin 1 of the IC socket.

Recheck connections and dress wiring away from the 28-pin header's pins because you probably won't get a second chance at this step once the IC socket's pins are completely soldered. Place the IC socket firmly atop the 28-pin header matching the IC socket's pins with the header's pins. Start by soldering two pins on each side of the assembly. Make adjustments in the alignment of the socket and header by reheating the four connections until everything looks good. Verify that the five connections (pins 1, 20, 26, 27, and 28) on the socket and header will not be shorting. If all is well, solder the remaining connections all the way around the socket/header assembly.

Insert the appropriate shorting jumpers on the configuration header. Cut a small piece (0.1 x 1 inch) of Mylar (or other insulator) and insert it into the battery holder, forming a "C"-shaped wall. This will prevent the coin battery from shorting to one (or more) of the header/socket pins. The coin battery can be removed/inserted

#### SOURCES

Digi-Key Corp.  
701 Brooks Ave. South  
P.O. Box 677  
Thief River Falls, MN 56701-0677  
(800) 3444539

Jameco Electronics  
1355 Shoreway Rd.  
Belmont, CA 94002  
(415) 592-8097

Dallas Semiconductor Corp.  
4350 Beltwood Pkwy South  
Dallas, TX 75244-3219  
(214) 4500400

The DS1210 is available from:

Circuit Cellar Incorporated  
4 Park Street  
Suite 12  
Vernon, CT 06066

Send \$7.50 (check or money order)

even with the low-powered RAM installed, although, if you use only one battery, the data will be lost when the battery is removed.

#### IT'S UP TO YOU

Select the device type as RAM and the appropriate device size (8K/32K). Insert the module into your favorite system's static RAM space and fill the contents of the RAM with code or data. The device can now provide battery-backed storage for your data-logging application or can be reconfigured as ROM (write protected) and inserted into a ROM socket for ROM emulation. I'll bet a whole

slew of applications are popping into your head right about now. I don't know about you, but if I don't write my ideas down, they end up permanently in the bit bucket.. probably means my battery needs changing.+

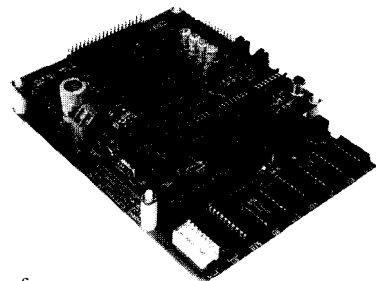
Jeff Bachiochi (pronounced "BAH-key-AH-key") is a member of the Circuit Cellar INK engineering staff. His background includes work in both the electronic engineering and manufacturing fields. In his spare time, Jeff enjoys his family, windsurfing, and pizza.

#### IRS

219 Very Useful  
220 Moderately Useful  
221 Not Useful

#### 16-BIT FORTH DEVELOPMENT SYSTEM

The development system consists of a two-board set. The target board can be used in a stand alone mode as a single chip unit with a FORTH kernel and up to 32K byte on-chip eeprom and 2K ram or with a piggy-back memory expansion board with either 64K bytes of 16 bit ram/rom memory or 32K bytes of a-bit ram memory.



#### MITSUBISHI M37700

The target board has two RS232/RS422 serial ports, sockets for 8 buffer IC's, two 40 pin headers for I/O or expansion, and battery backup for both the memory on the CPU and all of the expansion board ram.

The 16-bit single chip 7700 family has eight 16-bit timers, a watchdog timer, 68 I/O lines, two UARTS (synch or asynch), hardware multiply and divide, nineteen interrupts, and an 8-bit A-D converter with an 8 channel multiplexer, all with a typical power dissipation of 30 mW. They are available in both 8Mhz and 16 Mhz versions and with 512 to 2K bytes of on-chip ram and up to 32K bytes of on-chip ROM or EPROM.

Also available is a very low cost (\$125) prom programmer that can be used with the development system to burn either 27xx series of EPROM's or, with an adapter, the eeprom version of the 7700 chips.

Full development systems with FORTH source code for assembler, disassembler, editor, prom programmer and may other utilities as well as a 6K FORTH kernel in rom are available NOW!

Target Board prices start at \$200.00. Package prices and quantity discounts available also.

HORNE ELECTRONICS, Inc.  
33122 181st. Ave. S.E.  
Auburn, Wa. 98002  
(206) 735-0790

# CIRCUIT CELLAR INK ADVERTISER'S INDEX

Reader Service Number	Advertiser	Page Number
101	All Electronics	15
102	Alpha Products	4
103	Andratech	51
104	Annapolis Microsystems	33
105	A.T. Barrett & Associates	86
106	Avocet	c2
107	Aware Electronics	87
108	Binary Technologies	51
109	<b>Bitwise</b> Designs	c3
110	Byte Boss Intelligent Systems	87
111	Cabbage Cases	69
112	CAD Software, Inc.	23
113	Catenary Systems	43
•	Ciarcia Design Works	85
<b>114</b>	Ciarcia's Circuit Cellar	72
115	Circuit Cellar	61
116	Circuit Cellar	61
117	Computer Doctors	21
118	Computerwise	11
119	Control Devices, Inc.	14
120	Cottage Resources	32
121	Covox Inc.	80
•	Deus Ex <b>Machina</b> Engineering	86
123	<b>Dycor</b> Industrial (DA/M)	60
124	Electronic Energy Control	64
125	Emerald Microware	7
•	Epoch Data	86
<b>126</b>	Express Circuits	45
127	F&W Communications	59
128	Gott Electronics	52
129	Grammar Engine	72
130	GTEK	c4
131	<b>HathawayHite</b> Company	87
132	Hazelwood	10
133	High Res Technologies	87
134	<b>HiTech</b> Equipment Corp.	47
182	Horne Electronics, Inc.	71
135	Infinico	30
136	Information Modes	86
137	Innotec	14
138	Integrated Vessel	16
139	<b>Introl</b> Corp	37
140	Kelvin Electronics	87
141	Kelvin Electronics	87
142	Komputerwerk	86
143	Herb <b>Kraft</b> (R. Fringe Publishers)	16
144	Laboratory Microsystems	69
145	Link Computer Graphics	63
146	Logical Systems	80
147	Louis E. Wheeler	87
148	Master Voice	27
149	Meredith Instruments	11
150	Merrimack Valley Systems	11
151	<b>Micon</b> Corporation	50
152	Micro Dialects, Inc.	81
153	Micro Digital	78
154	Micromint	17
155	Micromint	33
156	Micromint	67
122	Micromint	77
157	Micro Resources	78
158	Ming Engineering (Electronics 1 2 3)	16
159	Needham's Electronics	70
160	<b>NeuralWare</b> , Inc.	22
161	Nisley Micro Engineering	86
162	NOHAU Corp	6
163	Paradigm Systems	63
164	Parallax, Inc.	42
165	PC Boards	6
166	<b>PseudoCorp</b>	49
167	R4 Systems	87
168	Real <b>Time</b> Devices	20
169	Silicon Alley	62
170	Softools	25
171	Software Success	87
172	Sunnyside Solar	70
173	Synetic Systems	58
174	Tanner Electronics	33
175	<b>TARDIS</b>	39
•	Tinney	39
<b>176</b>	Traxel Laboratories, Inc.	15
177	Universal Cross Assemblers	77
178	Unkel Software, Inc.	55
179	URDA, Inc.	86
180	Velotec	86
181	<b>Witts</b> Associates	86

## SUBSCRIPTION PROBLEMS?

If you HAVE PROBLEMS WITH YOUR SUBSCRIPTION (delayed OR MISSING ISSUES, CHANGE OF ADDRESS, OR QUESTIONS ON RENEWALS), CALL THE CIRCUIT CELLAR INK SUBSCRIBER SERVICE LINE AT (215) **630-1914** OR WRITE:

**Circuit Cellar INK**  
**Subscriber Service Department**  
**P.O. Box 3050-C**  
**Southeastern, PA 19398**

## IRS

### INK Rating Service

How useful is this article?

At the end of each article and some features there are **3-digit numbers** by which you can rate the article or feature.

Please take **the** time to let us, at Circuit Cellar INK, know how you **feel** our material rates **with** you. Just circle the numbers on the **attached** card.

# SILICON UPDATE

Tom Cantrell

## Old 8051s Never Die— They Just Get Smarter

### *New Power for a Controller Mainstay*

The world of **fashion** is known for styles that change? with the season (stock market? phases of the moon?). Change, whether worthwhile or gratuitous, is what drives consumers to trash last year's perfectly good outfits in favor of new. Despite rational pretensions, it seems like the world of high-tech gets equally **dizzy over** the latest fad. A voracious pack of headline-hungry trade press descend on the premier high-tech shows (Comdex, ISSCC, etc.) to report the latest trends ("Cycle Times Are Down, But Tiny Instruction Sets Put Pipelines On Display").

Detect a bit of cynicism? You can confirm it by taking a look in my closet. For office work there's the standard selection of red-white-blue office gear-ties and lapels all of medium width and nary a double-breast or sharkskin in site. Otherwise it's a nondescript mix of casual **styles**—jeans, T-shirts, sneakers—most notable for its lack of **notability**. In the back of the closet you can see the clothes I wore in college—I still like them and will wear them again when I shrink. I never throw any clothes out, though the most decrepit (I prefer "broken in") sometimes seem to disappear of their own volition. I suspect my wife must have a

hand in it-oil-stained **Apple sweatshirts** with ripped-off (unevenly, of course) sleeves may stand up by themselves, and even crawl, but they don't just disappear.

Needless to say, my taste in micros also leans toward the tried and true. Sure, I follow the news about the latest in high silicon couture—but when it comes to choosing my own "dailyware" I pretty much stick to the basics. Take the 8051 for example, a micro that is been around for a long time. Is it destined for the ash-heap of micro history or, like a good pair of jeans, does it get better with age?

#### MICRO 101

Before describing some of the latest developments in the 8051 world, I'll bring our younger readers (or those who have been on another planet for the last 10 years) up to speed with a short but sweet description of the device..

The 8051 (Figure 1) was designed as a follow-on to the world's first single-chip micro, the 8048. Compared to the 8048, the **8051's** main advantages are more complex (complete?) instruction set, greater external expandability (memory and I/O chips), and additional on-chip I/O.

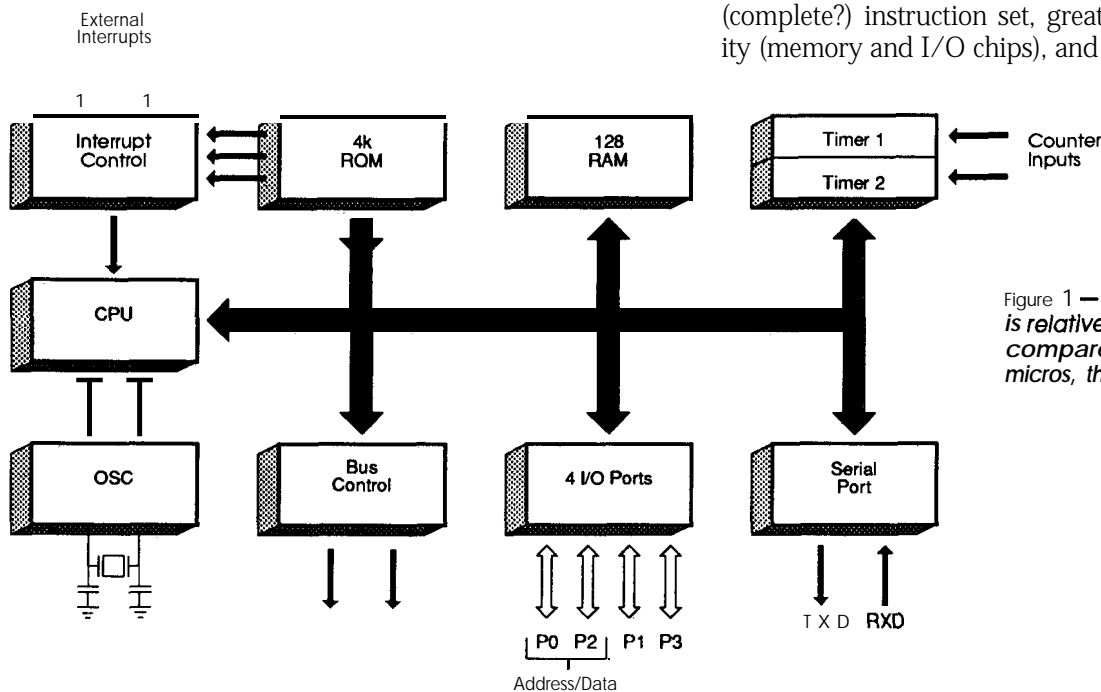


Figure 1—The 8051 CPU architecture is relatively straightforward when compared to Intel's other famous micros, the 8080 and 80x86.

Without getting into the gory details, I'll just say the 8051 CPU architecture—instruction set, registers, addressing modes, etc.—is relatively straightforward when compared to Intel's other famous micros, the 8080 and 80x86. In contrast to these, and other non-single-chip microprocessors, the 8051 is slightly less suitable for multichip, data processing oriented systems and is better suited to efficiently managing bit/byte-oriented control tasks. Notable advantages for the 8051 include tight instruction encoding (most instructions are one or two bytes long **and** execute in one machine cycle [**1µs** at 12 MHz]) and quick and easy bit handling operations (set, clear, test, etc.).

For larger applications, the 8051 offers external memory and I/O expansion via separate **64K-byte** code and data spaces (designers often map them to a single unified **64K-byte** code and data space). This is a big plus over the 8048 which features a **whopping 4K-byte** external address space. The single-chip orientation of the 8051 shows in the expansion bus; you won't find big-system features like DMA, vectored interrupts, DRAM refresh, or wait-state capability. Nevertheless, the 8051 can easily connect to the typical small-system mix of peripheral and byte-wide memory ICs.

On-chip I/O-wise the 8051 offers two **16-bit** timer/counters (they can be configured as one **16-bit** timer/counter, one **8-bit** timer/counter, and one **8-bit** timer) and a full-duplex UART. These modules are pretty basic, but many applications don't need a lot of bells and whistles. Also, the chip offers a total of 32 bits of parallel I/O (four bit-addressable **8-bit** ports—just remember that the pins are shared with other functions. Using the expansion bus and peripheral (e.g., UART, timer, and so on) I/O lines can

consume 24 of the 32 lines. Five interrupt sources (the timers, UART, and two external interrupt request lines) are serviced with a two-level programmable priority scheme.

Of course, the *raison d'être* for a single-chip micro is on-chip memory. The 8051 offers **4K** bytes of masked ROM and 128 bytes of RAM. Since masked ROM is only cost effective for highest-volume applications, two variants of the 8051 are offered to help the small guy. The 8751 replaces the 4K bytes of ROM with EPROM—the ideal solution, but expensive. Meanwhile, the 8031 deletes the ROM altogether, requiring the use of an external EPROM—a low-cost solution, but remember that I/O lines are lost. (I suspect the 8031 actually has a ROM inside. I wonder what the code is? Could be some bad ROM bits, endless NOPs, a factory test program—or perhaps the machine-readable legacy of **FlyByNightCo's** over-ambitious product plans.)

To me, the biggest advantage of the 8051 family isn't really technical; many **8-bit** micros can perform equally well. The real virtue is that the 8051, developed in an era with fewer lawsuits, has become a de-facto "people's micro." The list of suppliers, besides Intel, includes AMD, Harris, OKI, Siemens, and Signetics. Multiple vendors keep prices down and innovation up.

#### NEW TRICKS

Speaking of competition and innovation, let's take a look at a modern '51 family offering: the **83C522/87C522** (ROM/EPROM) (Figure 2) from **Signetics** (a subsidiary of Dutch electronics giant Philips).

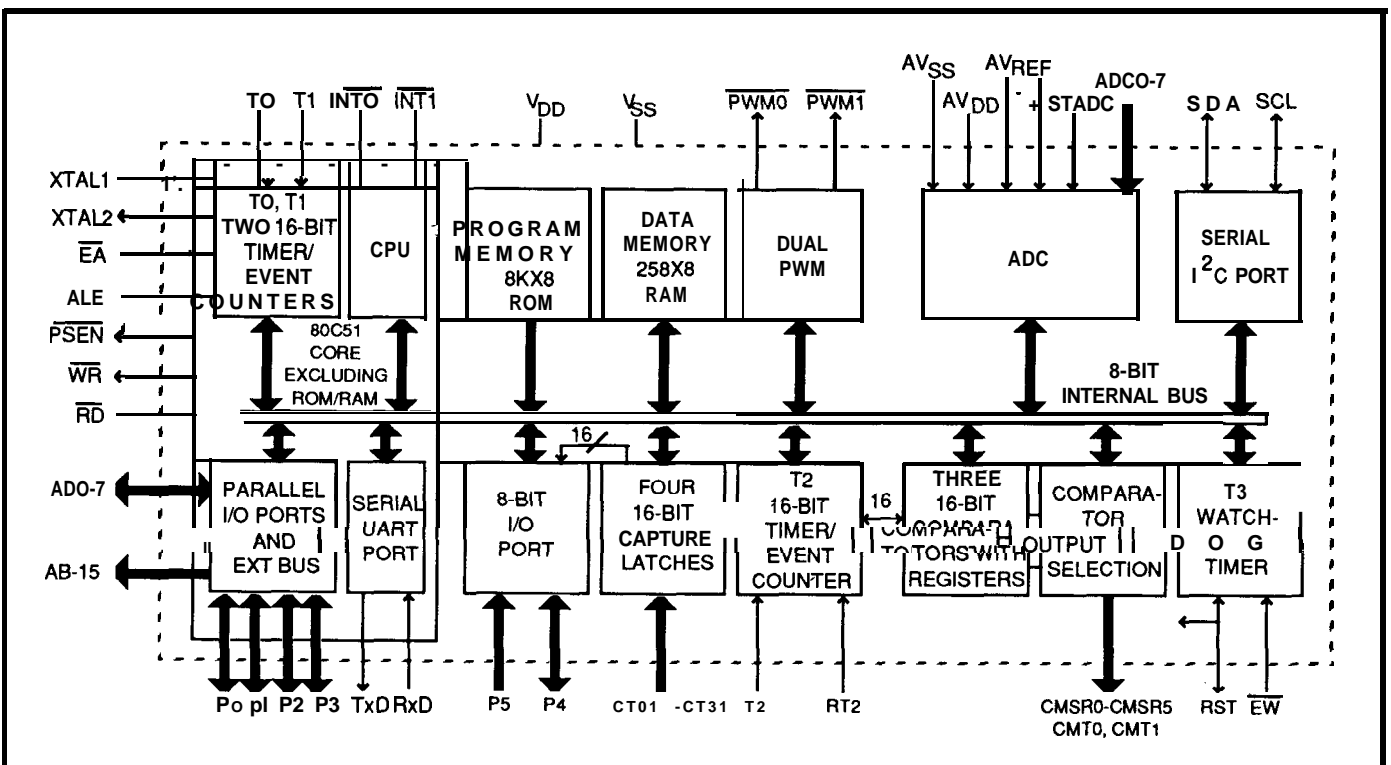


Figure 2—The 83C522/87C522 from Signetics is a superset of the 8051 offering such on-board I/O as ADC, PC port, and watchdog timer.



Of course, the "C" in the part number highlights one dramatic change over the decade: the shift from NMOS to CMOS. The '552, a much more powerful chip than the '51 as you'll see, consumes one quarter of the power of the older CPU (30mA vs. 125 mA max @ 12 MHz). Even better, '552 low-power modes (IDLE [7 mA] and POWER DOWN [7 μA]) can dramatically reduce average power consumption. CMOS is another example of high-tech giving us more for less.. **refreshing** isn't it.

Another basic change is the doubling of on-chip memory to 8K bytes (EP)ROM and 256 bytes of RAM. Larger applications will be able to fit on-chip without resorting to external memory. Presumably, future versions are on the drawing board with even more **memory**—increasing application complexity (and the hope to use "C") call for two, or even four, times as much. Nevertheless, a good assembly language programmer can do a heck of a lot with 8K bytes.

Don't forget the packaging advances in the last few years. The switch to high-density chip carriers packs the '552's 68 pins of functionality into less board space than the original 8051 40-pin DIP.

The I/O functions associated with the extra 28 pins are what really separate the old from the new. These upgrades include an 8-channel 10-bit ADC; an additional, and much smarter, 16-bit timer/counter; a 2-channel pulse-width modulator (PWM); an I<sup>2</sup>C (InterIC) serial bus channel; and a watchdog timer. We're talking serious system integration here.

The A/D converter (Figure 3) specs are pretty typical. Since it is a successive approximation type, conversion (initiated by software or an external trigger) takes a relatively long 50 machine cycles (50 μs @ 12 MHz), but that's certainly fast enough for most mechanical control tasks. The real advantage compared to an outside ADC IC is you don't have to resort to expanded bus mode (and lose a bunch of I/O lines) to see if the water is boiling.

The watchdog timer works as expected. You program a desired interval (2-510 ms @ 12 MHz, i.e., how long your system can remain crashed before the bits hit the fan). Then you partition/schedule your software to reload the watchdog within the specified interval. This can be tricky. You have to account for all program paths, interrupts, and so on. If the watchdog isn't reloaded in time, it resets the CPU (and can even drive the RESET pin as an output to reset external chips). An external ENABLE WATCHDOG pin and special timer reload protocol harden the watchdog against inadvertent disability.

Pulse-width modulation (PWM) simply means generating a pulse train whose interval and duty cycle is variable. Two independent channels, each with its own pulse output pin, are provided (Figure 4). The pulse frequency can be set from 92 Hz to 23.5 kHz at 12 MHz, with the duty cycle variable between 0% and 100% with resolution (8 bits) of better than 1/2%. PWMs are especially useful for controlling DC motor speed, but you can also use them to implement an integrating DAC. Otherwise, use them as regular outputs (0%=LOW, 100%=HIGH)—

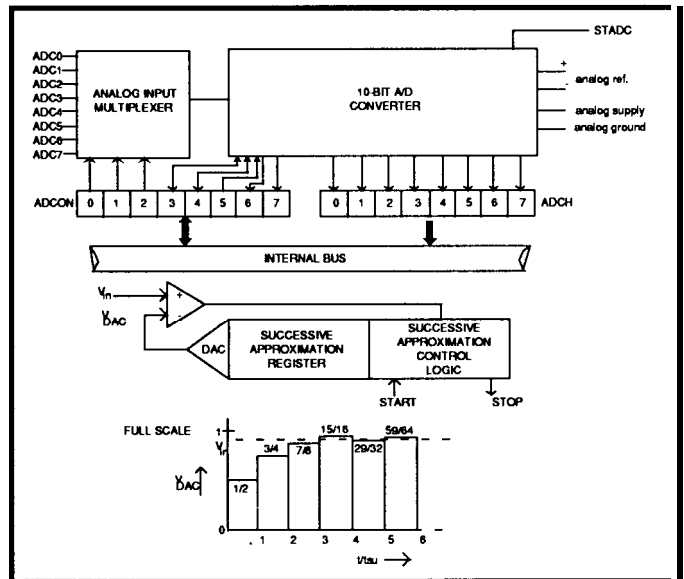


Figure J—The successive approximation ADC on the '552 takes 50 machine cycles to do a conversion.

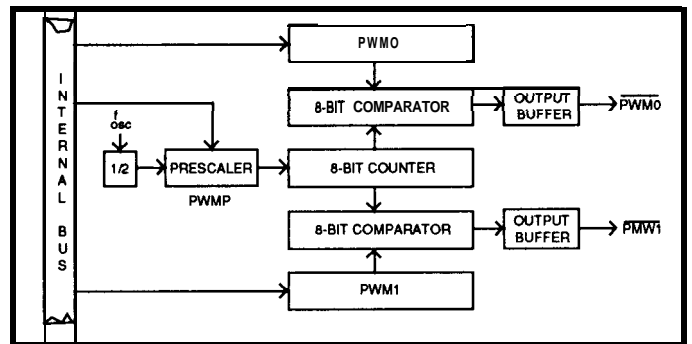


Figure 4—Two independent PWM channels generate pulse trains whose intervals and duty cycles are variable.

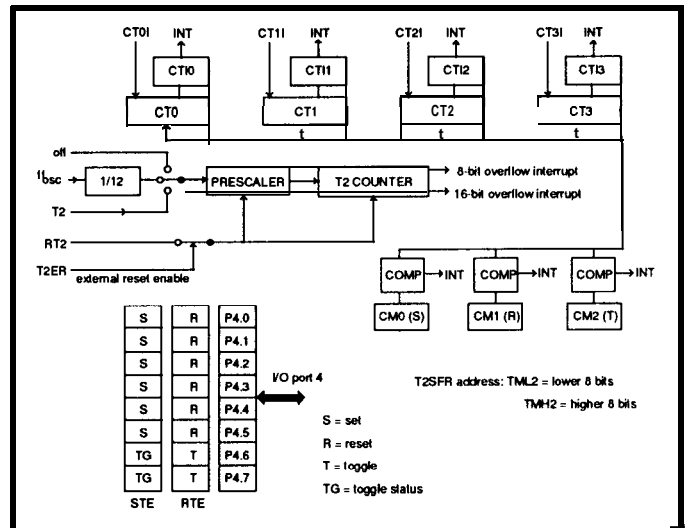


Figure 5—The new T2 16-bit counter/timer makes it easy to keep track of the real time between external events.

just the thing when the marketing department begs for an extra LED at the last second.

The new 16-bit timer/counter, T2 (Figure 5), embodies much of the '552's increased functionality. Actually, the timer/counter itself is much like the 8051's original T0 and

T1 timers (also included on the '552). The only major difference is that instead of being "loaded" by software, T2 can only be cleared by an external input. It's the input capture and output compare modules, supported with 12 I/O lines, surrounding T2 that add function. Four input capture lines are provided. They are just like external interrupt request lines, except a rising/falling edge (programmable) on any line also copies the T2 count to a register associated with each line. Thus, it's quick and easy to keep track of the real time between external events: it's just the difference between counts in each capture register. Meanwhile, eight output compare lines work in kind of an opposite fashion. They generate output events (set/reset/toggle) depending on the count in T2. Three registers are compared with T2 on every count and the preprogrammed events are triggered when there is a match. Variable phase shifting of the outputs is as easy as fiddling with the three registers-ideal for an application like engine timing and control (Figure 6).

The last major addition is the I<sup>2</sup>C port, which can best be described as a "LAN In A Box" (Figure 7). The idea of a low-cost interchip serial bus is really neat. I expect it will only grow in popularity. After all, why give up all your valuable I/O lines just to add a chip or two. Sure, the I<sup>2</sup>C 100-kHz maximum bit rate (<10k bytes/second considering protocol overhead) won't get a second look from the techno-gurus, but it is fast enough for lots of stuff and only uses two wires (clock and data). In fact, the patented I<sup>2</sup>C

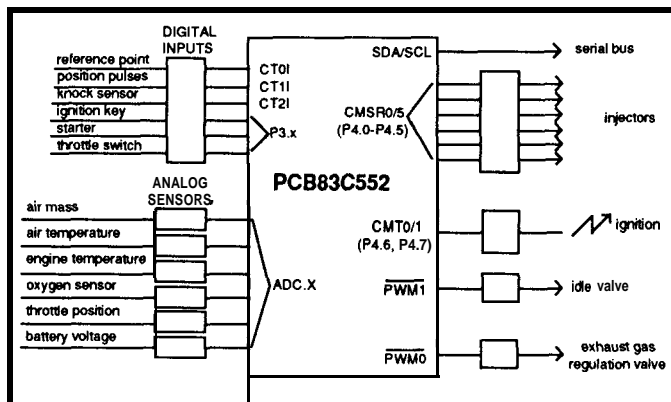


Figure 6—The I/O featured on the '552 makes it ideal for use in such applications as an engine controller.

protocol is more whizzy than it appears at first glance, featuring multimaster arbitration, adaptive data rates (accommodates fast and slow devices), and so on. Indeed, it takes a 100 or so lines of assembly language to drive the port—fortunately the listings are in the '552 user manual.

HAPPY BIRTHDAY

Though the original 8051 is aging, the plethora of suppliers ensure that innovation and specialization will continue. Signetics offers half a dozen variants in addition to the '552 described here—versions with-on-chip EEPROM,

## Cross-16 V2.0 Meta Assembler

Table based absolute cross-assembler using the manufacturer's assembly mnemonics.

Includes manual and MS-DOS assembler disk with tables for **all of the following processors:**

1802	64180	65C02	6801
6805	6809	68HC11	COP400
COP800	8048	8051	8085
8096	320C1X	TMS370	SUPER8
Z8	Z80	Z180	MORE...

**Users can create tables for other processors!**

**Generates listing, symbol table and binary, Intel, or Motorola hexcode.**

Free worldwide **airmail shipping and handling.**

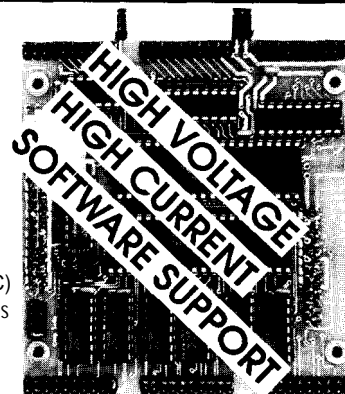
Check, Money Order or P.O. **US\$99.00**  
VISA, Mastercard and Canada **CN\$119.00**

Universal Cross-Assemblers  
POB 6158, Saint John, NB  
Canada **E2L 4R6**  
Voice/Fax: **(506)847-0681**

Reader Service #177

## Buffered high-voltage digital I/O for the RTC Family

### RTC-BUFIO



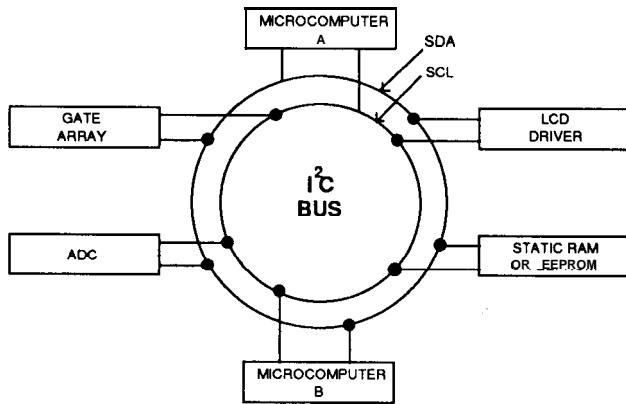
- 48 Digital I/O Lines
- 24 Digital Input Lines (Capable of withstanding  $\pm 30$  VDC)
- 24 Digital Output Lines (Capable of sinking 250 mA @ 50 VDC)
- 3.5" x 3.8" RTC form-factor
- BCC Bus-compatible (with BCC55R)
- RTC31 software supported with Micromint's new Threaded Programmable Controller Language

Priced at \$99 (Single quantity)



**MICROMINT, INC.**  
4 PARK STREET, VERNON, CT 06066  
CALL TO ORDER 1-800-635-3355  
(203) 871-6170  
FAX: (203) 872-2204

Reader Service #122



- PCF8566:** 96-segment LCD driver; 1:1-1:4 Mux
- PCF8576:** 160-segment LCD driver; 1:1-1:4 Mux
- PCF8574/A:** 8-bit remote I/O port (I²C bus to parallel converter)
- PCF8584:** 8-bit parallel to I²C converter
- SAA1064:** 4-digit LED driver
- SAA1300:** 5-bit high-current driver
- PCF8591:** 4-channel, 8-bit Mux; ADC + one DAC
- TDA8442:** Quad 6-bit DAC
- PCF8570/C:** 256-byte static RAM
- PCF9571:** 128-byte static RAM
- PCF8582A:** 256-byte EEPROM
- PCF8583:** 256-byte RAM/clock/calendar
- PCF8200:** Voice synthesizer (male/female speech)
- PCD3311/12:** Tone generator (DTMF/modem/musical)
- SCC68070:** 68000 CPU/MMU/UART/DMA/timer
- S80C552:** ROM-less version of S83C552
- S80C652:** ROM-less version of S83C652
- S83C552:** 256-byte RAM/8K ROM/ADC/UART
- S83C652:** 245-byte RAM/8K ROM
- S83C751:** 64-byte RAM/RK ROM

Figure 7-The I²C bus makes it easy to connect numerous devices together using just two wires. A sample of the chips available which support I²C is shown above.

ones shrunk into 24-pin packages, and so on. With all of the suppliers proliferating parts madly, we're facing a veritable baby boom of new '51s. So, let's celebrate the tenth birthday of the 8051—and wish it many happy RETs.✚

**Contact:**

Signetics Corp.  
811 E. Arques Ave.  
P.O. Box 3409  
Sunnyvale, CA 94088-3409  
(408) 991-2000

The 100-piece pricing for the '552:

- 80C552 (ROMless)—\$9**
  - 87C552 (Plastic PLCC—One Time Program)—\$25**
  - 87C552 (Ceramic windowed)—\$140**
- (note: a lower cost package is being developed)

*Tom Cantrell holds a B.A. in economics and an M.B.A. from UCLA. He owns and operates Microfuture Inc., and has been in Silicon Valley for ten years involved in chip, board, and system design and marketing.*

**IRS**

- 222 Very Useful
- 223 Moderately Useful
- 224 Not Useful

**LEARN MULTITASKING**



**COLLEGE KIT \$95**

The **smx** College Kit allows you to learn and to experiment with multitasking. It includes demo source code, a tutorial **User's Guide**, a **Reference Manual**, and help files - everything you need to try out multitasking! The CK is a reduced version of our **simple multitasking executive**. It runs at full speed and is ROM'able. Use it with **MS-DOS** or stand-alone. Works with the **80x86** family and **Microsoft** or **Turbo C** and assembler. The full version of **smx** is available for \$1995 with no royalties.

**MICROpd DIGITAL**

**1-800-366-2491**  
CYPRESS, CA 906304630

**68000**

**68000  
Debug  
Monitor**

- Suitable for embedded installation
- Off-the-shelf for any hardware
- Locate anywhere in memory
- Full-feature memory and register commands
- Breakpoints and tracing
- Interactive assembly and disassembly

- Absolute cross assembler
- Linking cross assembler
- C cross compiler
- Design and reference material

**68000  
Soft Tools**

**Custom  
Development**

- Hardware and software engineering design and prototyping available.



717-524-7390 or

60 SOUTH EIGHTH STREET LEWISBURG, PA 17837 717-523-0777

# Getting to Know You

*A New Feature Begins*

## PRACTICAL ALGORITHMS

*Scott Robert Ladd*

When inaugurating a new column, I always like to introduce myself and the “theme” I’ll be following. In the former case, I’m the former C columnist for *Micro Computer* magazine—alas, now sadly defunct—and a full-time writer. I focus on writing books and articles which are helpful to programmers who are trying to **produce** reliable and efficient programs.

Outside the world of computers, I’m backpacker, amateur astronomer, photographer, and father to my year-old daughter Elora. I live in Gunnison Colorado, a community of 6000 people located in the central Rocky Mountains. Parts of these columns get written while resting in an Aspen forest at 9000 feet above sea level. I’m not a radical environmentalist, but I am involved in water issues and conservation.

My computing environment is reasonably sophisticated. I use a 20-MHz **i386-based** computer, with a Mylex motherboard, for my primary MS-DOS work. This computer, known as Bull for arcane reasons, is equipped with 4 megabytes of memory, a 65-megabyte Mitsubishi MR535 hard drive, **16-bit VGA color** graphics, and **various peripheral** devices. I run MS-DOS 3.3 currently, since **MS-DOS 4.0x** isn’t quite compatible with everything.

In fact, this column is about the skill of the programmer. The focus is on algorithms, the step-by-step procedures used to accomplish a programming task. A program can be defined as a large algorithm built from smaller algorithms; the algorithms selected for a task will affect how well the program’s task is accomplished. Like a good mechanic, the programmer must know which algorithmic tool to use to best solve a given problem. My goal is to help you fill your toolbox with algorithms, and to show you how to select those algorithms.

I’m well-known in some circles as a C and C++ programmer; my first two books are on C++, and I’m now writing the C programming column for the magazine *Tech Specialist*. I’m not a language elitist, though. I happily create software using **80x86** assembly language, Turbo Pascal, Modula-2, and even FORTRAN. My philosophy is that the language used to create a program is less important than the skill of the programmer. C is not the best language for teaching; its syntax can be very obtuse at times, even for a C expert. C++ is also out; it is my language of choice, but is even more complex than its father, C.

Turbo Pascal is a structured language, but it is completely nonportable outside of the MS-DOS world. FORTRAN is old and klunky, and somewhat uncommon these days.

Modula-2 suddenly becomes the language of choice for this column. It has a clear syntax, is available on many platforms ranging from **CP/M** to UNIX, and is well-defined. I’ve seen Modula-2 used for every conceivable project from sophisticated databases to embedded micro-controllers. Designed by Niklaus Wirth as the successor to his earlier programming language, Pascal, Modula-2 has a “neater” syntax than Pascal and has extensions which make it a powerful tool for the professional programmer. Since the idea here is to present algorithms—and not language-specific constructs—Modula-2 will do nicely.

If you don’t already use Modula-2, there are some resources you can obtain. The language is defined by Wirth’s book *Programming in Modula-2*, printed by Springer-Verlag (ISBN 0-387-15078-1). There are several good Modula-2 books on the market; you can look for these at your local “techy” bookstore. Modula-2 is a small language, and many colleges and universities use it in their programming classes. Academic book stores will probably have resources on Modula-2.

For MS-DOS, the best Modula-2 compiler is from Stony Brook. It has a very fast and efficient code generator which easily produces the fastest programs I’ve seen from a high-level language. Also available for MS-DOS is the Jensen and Partners’ Modula-2 compiler. There are one or two Modula-2 compilers for **CP/M**, and I know that Modula-2 compilers for embedded systems exist. If you want a Modula-2 compiler, and don’t know where to find one, drop me a note and I’ll try to help you out.

### OPTIMIZING

Code optimizers are all the rage today. In essence, an optimizing compiler attempts to produce the fastest possible program from the source code you give it. Nearly **every** compiler advertisement contains a list of benchmark results, trying to prove that compiler A produces better code than compiler B.

Don’t be taken by the hype; the best code optimizer you have contains billions of interacting neurons and resides in your head. An optimizer may improve code

performance by 10 to twenty percent; your brain can increase program performance by orders of magnitude. No optimizer can convert a bubble sort into a QuickSort—but your brain can. The algorithms you implement, and the efficiency with which you program them, will determine the speed and size of your programs.

A case in point: Most programming texts will talk about how recursion is the natural way of programming certain algorithms. Recursion occurs when an algorithm is defined in terms of itself. In programming terms, a recursive algorithm is one in which a procedure calls itself in stages to accomplish a task.

A factorial is a value which is used primarily in statistics. The calculation of a factorial can be defined in terms of recursion. The computation of the factorial for an integral number takes this form:

$$\text{factorial } n = n * (n-1) * (n-2) * \dots * 2 * 1$$

In other words, a factorial is calculated by multiplying together all of the integers less than or equal to the operand and greater than or equal to one. The factorial of 5, for example, is  $5 * 4 * 3 * 2 * 1$ , or 120. The factorial of 1 is 1. Usually, the exclamation point is used to indicate a factorial calculation in mathematical texts;  $5!$  would mean the factorial of 5.

The simplest implementation of a procedure to calculate the factorial of a number would look like this:

```
PROCEDURE RecFactorial(n : LONGREAL) : LONGREAL;
BEGIN
  IF n = 0.0 THEN
    RETURN 1.0
  ELSE
    RETURN n * RecFactorial(n - 1.0)
  END
END RecFactorial;
```

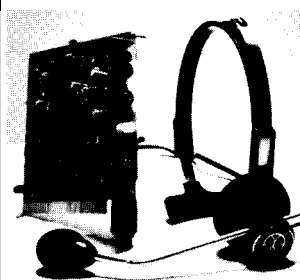
A factorial for a number can be defined as multiplying the number by the factorial of the number one less than it. The factorial of 5, for instance, can be defined as  $5 * 4!$ . The factorial of 4 can be defined as  $4 * 3!$ , and so on. So, a recursive algorithm-like the one above-calls itself to calculate the factorial of each successively smaller number, until it reaches 1. RecFactorial can be called recursively because it is self-contained; the only variable it uses is n, which is local to each call to RecFactorial. Procedures which modify global variables cannot (in general) be made recursive. The recursive calls end when n is 1; RecFactorial can at that point return a constant value of 1 rather than calling itself further.

The recursive version of the factorial procedure is clear and to the point—but is it efficient? The answer is no; recursion involves making repeated function calls which increase program overhead. For every call to RecFactorial, the value of n must be pushed onto the stack, along with a return address, when the call is made. When an invocation of RecFactorial is done, these values must be popped off the stack. This pushing, calling, and pop-

## TALK TO YOUR COMPUTER

WITH VOICE MASTER KEY® FOR PCs/COMPATIBLES  
VOICE RECOGNITION WITH SPEECH RESPONSE

GIVE A NEW DIMENSION TO PERSONAL COMPUTING The amazing Voice Master Key System adds voice recognition to just about any program or application. Voice command up to 256 keyboard macros from within CAD, DTP, word processing, spread sheet, or game programs. Fully TSR and occupies less than 64K. Instant response time and high recognition accuracy. A real productivity enhancer!



**SPEECH RECORDING SOFTWARE**  
Digitally record your own speech, sound, or music to put into your own software programs. Software provides sampling rate variations, graphics-based editing, and data compression utilities. Create software sound files, voice memos, more. Send voice mail through LANs or modem. A superior speech/sound development tool.

**INTERACTIVE SPEECH INPUT/OUTPUT** Tag your own digitized speech files to voice recognition macros. Provides speech response to your spoken commands -- all from within your application software! Ideal for business, presentation, education, or entertainment programs you currently use.

Augment the system for wireless uses in robotics, factory process controls, home automation, new products, etc. Voice Master Key System does it all!

EVERYTHING INCLUDED Voice Master Key System consists of a plug-in card, durable lightweight microphone headset, software, and manual. Card fits any available slot. External ports consist of mic inputs and volume controlled output sockets. High quality throughout, easy and fun to use.

ONLY \$149.95 COMPLETE

ORDER HOTLINE: (503) 342-1271 Monday-Friday 6 AM to 5 PM Pacific Time. VISA/MasterCard phone or FAX orders accepted. No CODs. Personal checks subject to 3 week shipping delay. Specify computer type and disk format (3 1/2" or 5 1/4") when ordering. Add 55 shipping charge for delivery in USA and Canada. Foreign inquiries contact Covox for C & F quotes.

**30 DAY MONEY BACK GUARANTEE IF NOT COMPLETELY SATISFIED.**

CALL OR WRITE FOR FREE PRODUCT CATALOG.



**COVOX INC.**  
675 CONGER ST.  
EUGENE, OR 97402

TEL: (503) 342-i 271  
FAX: (503) 342-i 283

Reader Service X121

## SIM51 Ver 3.00

Assemble Load	Options	Breakpoints Processor	Sim51 Processor	Dump Registers	Editor	Go Save	History Unassemble	Interrupts exit
0044	MO	External Program	Disabled	PSW	---	0	83C451	Port 0/P
0045	IN	External Data	Disabled	D0 PSH	00	87	PCON	00 80 P0
0046	JZ	Port Source	Sim51	D0 ACC	00	100	Control	90 P1
0048	MO	Interrupt Sres	Sim51	F0 0	00	80	TCOM	00 A0 P2
0049	SI	Serial Input Src	Buffer	81 SP	07	89	TMOD	00 B0 P3
004C	MO	Serial Output To	Port 6 Input Src	82 DPL	6F	8A	TL0	00 C0 P4
0040	MO	Port 6 Input Src	Port 6 Output To	83 DPH	00	8B	TL1	00 C8 P5
0043	CL	Port 6 Output To	File	83 DPL	00	8C	TR0	00 D0 P6
0044	MOVC	A 00+DPTR	COM1	FC	00	8D	TH1	00 Port 1/P
0045	INC	DPTR	300	00 R0	00	TMR	0 0000	80 P0
0046	INC	0004CH	600	81 R1	00	TMR	1 0000	90 P1
0048	MOV	CON, A	1200	82 R2	00	Int	Control	A0 P2
0049	SJMP	00043H	2400	83 R3	00	A8	IE	00 B0 P3
004C	NOP		4800	04 R4	00	88	IP	00 C0 P4
			9600	05 R5	00	TIP	0	C0 P5
			19200	06 R6	00	Ser	Control	D0 P6
Hello	World!	My name is Mat		07 R7	00	98	SCON	00 P6
004C	NOP			08 OR0	00	99	SETP	00 D0
				09 OR1	00	99	S5FR	00 D0

- Simulator/Debugger for 8051, 8052 & 80451
- Full Screen, Menu driven
- ALL registers ALWAYS displayed
- Extensive Serial Port and '451 Port 6 support
- In circuit debug with BusBoy51 probe
- Debug all Memory models, Ext. & Internal
- Call or FAX for FREE demo version
- Download demo (D51 DEMO.EXE) from BBS (315) 475-3961 (2400/1200,8,N,1)
- Only \$250, with BusBoy51 \$499

**CALL (315) 478-0722 FAX (315) 475-8460**

Logical Systems Corporation  
P.O. Box 6184, Syracuse NY 13217 USA

Reader Service #146

ping slows down `RecFactorial` considerably.

**There's another problem with `RecFactorial`:** Every call to it uses 12 bytes of stack space for `n` and the return address. Most programs have a limited amount of stack space. The more arguments a recursive function requires, the more stack space it uses. Using `RecFactorial` to calculate `100!` will use 1200 bytes of stack space—clearly an unacceptable amount for many applications.

A procedure for calculating a factorial can be written without the use of recursion. For example.. .

```
PROCEDURE IterFactorial(n : LONGREAL) : LONGREAL;
VAR
  result : LONGREAL;
BEGIN
  result := 1.0;
  WHILE n > 1.0 DO
    result := result * n;
    n := n - 1.0;
  END;
  RETURN result;
END IterFactorial;
```

This function uses a **simple** loop to calculate a factorial. Without recursion, this function **uses** only 16 bytes of stack space (for `n` and `result`) no matter which factorial is being calculated. Lacking the overhead of multiple **function calls**, `IterFactorial` is **18% faster than `RecFactorial`**. Clearly, the better algorithm is the iterative one. An additional benefit is that `IterFactorial` performs one **less "loop"** than `RecFactorial`, because it can stop when `n` is reduced to 1. This eliminates an extra loop which would multiply result by 1.

Any algorithm which can be defined using recursion can also be defined using iteration. There's a long mathematical proof of this, but I'd prefer it if you'd just take my word for it. I have yet to find an algorithm which is not faster and smaller when defined iteratively as opposed to recursively.

## DESIGN VERSUS HACKING

I'm a believer in designing programs, as opposed to building them on the fly. Many programmers from the old "hacker" school (which I attended) prefer to build programs without doing **any** real design in the first place. This has led, in my view, to the current problem with code bloat: Programs which perform poorly while using **copious amounts of memory**. As software complexity increases, programmers need to use more and more thought in how their programs are designed.

A programmer, in my mind, has a responsibility to produce the most efficient possible program for a given situation. In a statistics package where thousands of factorials may be performed on large numbers, the 18% performance difference between a recursive factorial procedure and an iterative one may be vital. To build better programs, a programmer has to work smarter. You need to look **at** how your algorithms use system resources to determine if they are the best for a given situation. Don't make assumptions; more programs have ended up in the trash

heap because of poor design than any other reason.

In the next column, I'll dive into sorting problems. Every program sorts data, and it's a time-consuming task which requires the selection of the proper algorithm. I'll be showing you the oft-forgotten **HeapSort** algorithm, along with an iterative version of the **QuickSort** algorithm that runs 20% faster than the traditional recursive **QuickSort**.

## WRAPPING UP

I like feedback from you; having contact with the readers of my column was perhaps the best thing about writing for *Micro Cornucopia*. Feel free to drop me a line at the given address; please avoid calling me if you can, because I get far too many calls already. Send me your gripes, suggestions, and thoughts about where you want this column to go. I can't promise a response, but at least I can tell you I'll pay attention. After all, this column is written for you, not me!

Until next time.. ❀

*Scott Ladd is a writer specializing in computer software. Correspondence concerning "Practical Algorithms" may be sent to him at: Scott Robert Ladd, 705 West Virginia, Gunnison, CO 81230, (303) 641-6438*

## IRS

225 Very Useful  
226 Moderately Useful  
227 Not Useful

**NEW!!**  
**68000, COP800, PIC16xx**  
**versions!**

**µASM™**

**Cross Assemblers  
for the Macintosh™**

- TEXT EDITOR, CROSS ASSEMBLER, AND COMMUNICATIONS FACILITY IN A COMPLETE INTEGRATED DEVELOPMENT ENVIRONMENT
- MACROS
- CONDITIONAL ASSY
- LOCAL AUTO LABELS
- SYMBOL TABLE CROSS REF
- S OR HEX FILE OUTPUT DOWNLOADS TO MOST EPROM PROGRAMMERS

u.s. **\$149.95**  
**EACH**  
**PLUS S/H\***

AVAILABLE FOR MOST 8-BIT MICROPROCESSORS AND 680001010. CALL OR WRITE FOR TECHNICAL BULLETIN. 30 DAY MONEY BACK GUARANTEE. MC/V/AE.

\* PER SHIPMENT:  
\$4 CONTIGUOUS USA  
\$8.50 CANADA AK, HI  
\$15 INTERNATIONAL

**Micro Dialects, Inc.**  
DEPT. C, PO BOX 30014  
CINCINNATI, OH 45230  
**(513) 271-9100**

Order Service #152

# CONNEC- TIME

Conducted by  
Ken Davidson

## Excerpts from the Circuit Cellar BBS

The Circuit Cellar BBS  
300/1200/2400 bps  
24 hours/7 days a week  
(203) 871-1988  
Four incoming Lines  
Vernon, Connecticut

*Dale Nassar's article about computer-generated holograms in issue #14 of CIRCUIT CELLAR INK received a lot of positive feedback. In this issue's first discussion, a user of the BBS chats with Dale over his successes in making holograms with a Mac using Dale's article as the basis. We'll also be looking at word processors which support foreign users, and choosing replacement transistors.*

**Msg#:27843**

From: STEVE BUNCH To: DALE NASSAR

Dale, I enjoyed the article about computer-generated holograms a lot. It wasn't clear to me from the article, though, how you actually go about viewing the holograms after you have them on film. I have a He-Ne laser and took 35mm shots of both my Mac screen running a version of your program and shots of the magazine. The only image I'm getting is so small that it's a point (or else an artifact)—I can't see it! So how do you go about viewing it, and taking photographs of the virtual image?

FYI, I came across a paper (referenced in a holography book) about this subject: "Computer Holograms With a Desktop Calculator," by James S. Marsh and Richard C. Smith, in American Journal of Physics, Vol. 44, No. 8, Aug. 1976(!). In the article, they generate stick letters by directly evaluating the Fourier transforms of the line segments making up the letters at each point in the hologram plane. Seems complementary to your article.

**Msg#:27846**

From: DALE NASSAR To: STEVE BUNCH

To view the real image of the pattern of Photo 6 in the article, you should use an uncollimated laser beam. Place the hologram pattern far enough away from the laser so that the beam covers most of the pattern (around 20 feet for my laser) and the real image should be projected onto a screen (or whatever) about 15 feet beyond the hologram—hold a piece of paper in the beam's path and walk down with it until you get a good image. The image you see should be as good as Photo 5a.

The photos of the reconstructions in the article were not nearly as good as they could have been, but the schedule was very tight and I did not have time to reshoot them. These distances are so large because of the coarse structure of the pattern. To see the

virtual image, you have to really know what you are looking for and where to look—it is very small and distant but can be seen by looking through the film toward the laser. Because this is an offset hologram, the laser beam will just miss your eye.

I had originally planned to make holograms with a ultra high resolution plotter, but didn't have time. This would produce better virtual images.

I was really surprised at how good the holographic images came out—I think it is amazing that a flat monochrome pattern can produce a TRUE 3D image. Also, full-color holograms have been made by using three lasers (red, green, and blue). With my synthesizing method I should be able to create holograms that can be viewed in white light. This would be done by assigning continuous wavelengths using integration over the continuous spectrum (as compared to the summation of the three laser colors) thus simulating coherent white light (see the cover of that issue of Circuit Cellar INK).

**Msg#:27873**

From: STEVE BUNCH To: DALE NASSAR

Thanks. Will try it out tomorrow. Have in the meantime refined my Mac program. It now generates gray-scale pixels (128 levels). I'm looking forward to seeing what kind of difference this yields.

The paper I mentioned in the last message used a spread, then recollimated laser for readout, as near as I could tell (they were a bit vague), and referred to using a negative focal length lens to spread the beam to enlarge the virtual image. I've just started playing with this, so I'll let you know if I have any useful results.

**Msg#:27881**

From: DALE NASSAR To: STEVE BUNCH

In theory, the main reason to preserve a gray scale is to prevent higher order diffractions during reconstruction. It is undesirable for higher order (first, second, etc.) diffracted images to be superimposed on the primary (zero order) reconstructed image. Because I did not record a gray scale, I assumed that the higher order diffractions would appear in the reconstruction, so I used diffraction grating equations to determine minimum fringe spacing so that the main image is just separated from the others.

To my surprise, in the reconstruction the higher order diffractions were almost nonexistent even with the binary nature of the holograms (maybe because the "black" portions of the hologram were not completely opaque). So I would think that using a gray scale should have little effect on the reconstructions-but as I have learned in this project, never assume anything until you try it. I haven't tried this, so please let me know how it turns out.

In Listing 2 of the original article, the "trigger point" of the film is determined in line 110 ( $s=0$ ). Because just about any level can be assigned here, there are no unique patterns for any one subject—this is interesting especially when considering the coarseness of the pattern. I used 0 because it was the natural first choice based on amplitudes. You may want to try other values here. One extreme should yield a mostly opaque pattern and the other a mostly clear one. Also, the pattern (on the video display) produced by the gray-scale interference should be interesting.

Let me know what results you get.

**Msg#:27955**

From: STEVE BUNCH To: DALE NASSAR

Success finally. The real image was where it was supposed to be. I've been struggling with a balky laser (the home-made mount for the tube changes with temperature, and the tube goes out of alignment and fades) and aligning front-surface mirrors in the basement to get a long enough optical path to do it indoors. My one outdoor experiment last night was fun for the neighbors, but too much of a hassle. It's pretty impressive to see the image come into focus as you approach the focal point. I still haven't found a recognizable virtual image, though.

My images are not as clean as the ones in your article. Two of the lobes are quite good, the third is very noisy and buried in trash (the direct feed-through of the beam, at least partly). The entire field (at the focal point of the image) has a lot of noise in it. I made a couple of other shapes (circle of dots and a simple point). The point was fine, but I had too many dots in the circle, which made them hard to see. Part of the problem is probably that I didn't mask the edges accurately, so there was more light coming around the hologram than desired. I'll clean up the details a bit and see how much that helps.

My Macintosh program is working well. It allows you to save the raw pixel data at any point in the computation, and come back later to finish it. I put in the ability to generate the hologram using not just black-and-white (Fresnel) patterns, but gray scale ("Gabor") as well, where each point on the hologram can be any of 128 gray levels ranging linearly from black to white. Unfortunately, my experiments with it were pretty much a disaster. I got some artifacts which, if I knew exactly how to analyze them, would probably tell me exactly what was wrong with my monitor's linearity and my camera angle. I'm guessing that it's more critical to get every pixel exactly as it's supposed to be.

At any rate, more experimentation is called for. I'm displaying the images on a color monitor, and there may also be some effects from the shadow mask. (I was using a sharp macro lens and good resolution film [T-MAX].) It takes the program about 30 minutes to compute the figure in your article, at the same resolution, running on a Mac IIcx. The program is written in Think C, and

uses a public domain application skeleton called Transkel to handle the boring stuff with windows and such.

I just read your note (the above was written off-line). The gray scale patterns are in fact delightful to look at. I'll do some playing with the "trip point", too.

**Msg#:28051**

From: STEVE BUNCH To: DALE NASSAR

Tried again on the gray-scale holograms with much better results. Turns out I had taken the shots closer to the screen, so the focal distance out to the real image was longer and I didn't go out that far last time. Got some extremely nicely focused points—much clearer than I've seen so far. However, I have the same problem with both the b/w and gray versions: lots of undiffracted light (or at least unfocused by the hologram) feeding straight through to the field. The vast majority of my energy isn't going into the pattern at all, but is showing up as patterned noise.

I got a very noticeable improvement when I improved the alignment of the hologram, laser beam, and mirrors, but it was still not nearly as good as it could be (e.g., the gray-scale one had *\*really\** sharp dots—it was seven points in a circle-surrounding a rectangle of red muck.) I'm going to play some games with lenses to reduce the size of the playing field a bit and see if alignment or mirrors or something like that is causing trouble.

**Msg#:28093**

From: DALE NASSAR To: STEVE BUNCH

Sounds like you're reproducing the images as well as can be expected without additional optics. The reason I suggest viewing the offset hologram in the article is so that the straight-through beam will be separated from the image at the focal length. I think I oriented the rose so that the beam would pass between lobes if the separation between the image and the laser beam was small.

Although there is a theoretically ideal pattern size for a given set of parameters, I don't think it is very critical in practice (as you noted when you obtained a longer focal length with an oversized pattern)—you may want to try to get a smaller focal length by illuminating a reduced-scale pattern.

**Msg#:28111**

From: STEVE BUNCH To: DALE NASSAR

I have done a little more experimenting, but so far, the image seems to be about as good as it's going to get—as you said. I've had some unexplained nonresults which I need to work on some more (some patterns just aren't generating an image at anywhere near the correct focal plane, but similar ones do).

*The uncut version of this message thread would fake up twice as much space as we have room for here. For those interested in pursuing computer-generated holograms, the entire thread is available on-line on the Circuit Cellar BBS starting at message #27843 or on the Circuit Cellar BBS on Disk installment for issue #15 (not for this issue).*



---

*Most word processors work well with English, but what happens when the text doesn't even go from left to right?*

**Msg#:25541**

From: KI SUNG To: ALL USERS

I am looking **for an** Arabic and Farsi (Persian) word processor for my clients. If anyone knows where I can purchase one, please let me know. Thank you.

**Msg#:25557**

From: CURT FRANKLIN To: KI SUNG

I don't know if it qualifies as a true Arabic/Farsi word processor, but I am told that WordPerfect 5.0 is able to deal with Arabic/Farsi character sets. How easily it deals with them depends on which printer you're using. If you're coming up blank, it might be worth a try.

**Msg#:25577**

From: ED NISLEY To: CURT FRANKLIN

It deals with them character-by-character, which isn't what you want. There are several ads in the back pages of PC Magazine for word processors that handle all manner of non-European character sets, so a glance through there will be productive.

**Msg#:25884**

From: PELLERVO KASKINEN To: KI SUNG

Check for **Nota Bene**. It is supposed to handle alien characters and languages better than **any** domestic word processor. If I have understood correctly, it is a customized version of Xywrite, itself known of its abilities to handle all kinds of specials. I think **Nota Bene** supports even writing from right to left like the Arabic and Farsi are done.

---

*What happens when you have a circuit, a blown component, and no source for replacement? A transistor by any other name would still be a transistor--at least according to some people.. .*

**Msg#:26393**

From: EDWIN BIGIO To: ALL USERS

I'm having problems getting a transistor used by a Compaq portable power supply. It is made by Motorola and has the markings "539" followed by "1191." It is in position **Q3** and the schematics furnished by Sam's makes reference to it as a "508 NPN" transistor. Motorola told me the part was manufactured specially for Compaq and it is impossible for me to get one from them. Any help would be appreciated.

**Msg#:26488**

From: ED NISLEY To: EDWIN BIGIO

This may get me in a lot of trouble with the analog types, but I've always held that there are only four types of transistors: big/little and **NPN/PNP**. (Actually, there's a fifth category: anything to do with RF...but nobody understands how to make those work.)

Basically, you look at the defunct transistor and decide if it's big or little. Then you check the schematic to find out whether it's PNP or NPN. Then you go out and buy a new transistor to suit.

Actually, this isn't quite as cavalier an approach as you might think, because a good circuit design doesn't depend on the exact transistor parameters. As long as you get within the right ballpark, the circuit ought to work pretty well.

Remember that a "special part" may differ from the standard in, say, the lower temperature limit or a minimum DC current gain or something that a standard part will almost certainly satisfy.

You might want to start out with dummy loads and make sure that everything is sensible, but I bet it'll work just fine.

**Msg#:26534**

From: DAVE EWEN To: EDWIN BIGIO

Maybe you could find out what part other computer power supplies use, and then get one of those...of course make sure the schematic shows it as being the same type (**NPN, PNP**)

They don't use **FETs** in these supplies do they?

**Msg#:26588**

From: ERIC BOHLMAN To: ED NISLEY

While you're right that a good design doesn't depend on exact values for most transistor parameters (beta being the usual example), you still have to take voltage and current ratings into account and make sure that the replacement is rated at least as well as the old part.

**Msg#:26600**

From: ED NISLEY To: ERIC BOHLMAN

Sure. If you know anything about the part, you use what you know...but if you don't, that's when the rule of thumb comes in handy.

For instance, when the schematic shows a cap upstream of the transistor with a 400-VDC rating, you don't use a **2N4401**. On the other hand, if the cap is rated 15 VDC, hey, you give it a shot.

Or if the transistor is **one** of a matched set, **maybe** you think about getting transistors that come in matched pairs rather than two randomly chosen things from your junk box.

But if **all you've** got is a board with three holes, no schematic, and a strong desire to get it working...

**Msg#:26751**

From: PELLERVO KASKINEN To: ED NISLEY

Not that I would want to put you into any trouble, even though I am pretty much an "analog type," but I just thought I would make your prediction come a little closer. ;-)

In addition to the four basic "types" of unknown transistors, there are a couple of other concerns. You probably need to pay attention to the operating voltage; that is, there are low-voltage transistors, say below 100 V, and there are high-voltage transistors, maybe up to 350 V in smaller transistors or 500 V in larger ones.

Then there is the issue of technology: are they **FETs**, normal bipolar transistors, or possibly Darlington connected (two transistors internally connected for increased gain).

Finally, there **maybe SCRs** and **triacs** that look just like transistors but behave in a very different way.

That helped to clarify the muddy waters, did it? No, in principle I follow very much the same pattern of thought, if I need to look at some replacement issues. However, I am worried every time when the transistor or "transistor" is so shorted that I can not reasonably determine its original type. Well, I am also worried, if it appears to be open. I go to great lengths of effort trying to locate some info even about custom-labeled devices. Sometimes the substitution catalogs come to help.

One last thought, some of the transistors are not really custom. There are other standard marking systems than the JEDEC. The Japanese have their own (something like **2Sxxx**), the Europeans their own Pro Electron system (**BCxxx, BFxxx, BSxxx, BFXxx**, and so on) and of course the Russians have their own, but you are less likely to be stomping into those, I assume.

**Msg#:26946**

From: ED NISLEY To: PELLERVO KASKINEN

And, of course, it could be a little two-terminal regulator done up in a 3-lead package...

I guess I was assuming too much knowledge about the problem...like you really know that little three-legged corpse started out life as a bipolar transistor, or you could infer it from a casual glance at the surrounding circuitry. If you haven't the foggiest idea what it is, sticking a **2N4401** in the socket probably won't work too well.

But, hey, it's worth a shot!

**Msg#:26972**

From: MICHAEL COVINGTON To: ED NISLEY

He \*did\* say it was an NPN transistor.

Just find out the voltage ratings for the device (probably pretty easy to guess) and get one in a matching package and **pinout**, and you're there.

**Msg#:27085**

From: ED NISLEY To: MICHAEL COVINGTON

And, with a little ingenuity, you can make any of the six possible **pinouts fit...now** there's an opportunity for standardization!

The Circuit *Cellar* BBS runs on a **10-MHz Microminif OEM-286 IBM PC/AT-compatible computer using the multiline version of The Bread Board System (TBBS 2.1M)** and currently has four modems connected. We invite you to call and exchange ideas with other Circuit *Cellar* readers. It is available 24 hours a day and can be reached at (203) 871-1988. Set your modem for 8 data bits, 1 stop bit, and either 300, 1200, or 2400 bps.

**IRS**

- 228 Very Useful
- 229 Moderately Useful
- 230 Not Useful

**SOFTWARE and BBS AVAILABLE on DISK**

**Software on Disk**  
Software for the articles in this issue of Circuit *Cellar* INK may be downloaded free of charge from the Circuit *Cellar* BBS. For those unable to download files, they are also available on one **360K, 5.25" IBM PC-format** disk for only \$12.

**Circuit *Cellar* BBS on Disk**  
Every month, hundreds of information-filled messages are **posted** on the Circuit *Cellar* BBS by people from all walks of life. For those who can't log on as often as they'd like, the text of the public message areas is available on disk in two-month installments. Each installment **comes** on three **360K, 5.25" IBM PC-format** disks and costs just \$15. The installment for this issue of INK (**August/September 1990**) includes all **public** messages **posted** during May and June, 1990.

To order either Software on Disk or Circuit *Cellar* BBS on **Disk**, send check or money order to:  
Circuit *Cellar* INK — Software (or BBS) on Disk  
P.O. Box 772, Vernon, CT 06066

or use your **MasterCard** or Visa and call (203) **875-2199**. Be sure to specify the issue number of each disk you order.

**The Ciarcia Design Works**

Steve Ciarcia has assembled a team of engineers, designers and programmers to design and manufacture products ranging from the multiprocessor Mandelbrot engine to ROVER and the Serial EPROM Programmer. Now you can put the Ciarcia team to work for you.

Steve Ciarcia and his staff have design expertise in fields as diverse as fast video digitizing, control networks, **multiprocessor** design, and wireless communications. Current capabilities include every phase of design and production, from initial concept through product packaging.

Whether you need an on-time solution for a unique problem, **complete** support for a startup venture, or experienced design consulting for a Fortune 500 company, the Ciarcia Design Works stands ready to work with you.

**Remember...a Ciarcia design works!**

**Call 203/875-2199 Fax 203/872-2204**

## Flash or Splash?

**R**ather than bore you with prophetic statements about the computer industry, I thought I'd use my space this month to tell you what **I've been doing lately and** what you may or may not be seeing as a future project. The last time I left you I had finished adding a big solarium to the house. Presuming that such an environment should be a suitable candidate for complete instrumentation and control, I had even put in about a thousand feet of cabling during construction. Of course, as my data logger proved beyond doubt, the six feet of wire from a dumb fan thermostat was all that was required for 99% satisfactory environmental control.

Setting out to employ computer control and then not using it does not mean failure, however. Successful **engineering** really means collecting **data** and then applying the proper solution. Of course, I did have a few trepidations when, after months of telling Curt that I'd have a controller project for that issue, I had to tell him that I "engineered" it out! Fortunately, everyone understands that my credibility, like this magazine's, is built on real situations and real applications and that's why I told it like it was.

I'm presently at the crossroads of two other projects that Curt expects me to document. Here again, I have a pair of situations which could either be described as elaborate engineering opportunities or overkill, depending upon whether, and to what degree, it all really works.

One project is lightning protection. As most of you probably know, my house seems to be ground zero for thunderstorm activity in Connecticut. While probably nothing compared to the boomers in Florida or Indiana, I still end up with damage claims averaging \$5000 a year. At the end of last year's bout which left a scorch on the garage ceiling and a freon-filled Circuit Cellar, I decided to approach solving it like an engineer.

After trenching hundreds of feet of **1/2"** stranded copper cable around the property, wiring a web of cables and lightning rods on the roofs, and connecting practically

every piece of metal in the house to this virtual ground plane, I sit waiting for Zeus to throw his next bolt so I can see if this whole mess works. As of this writing (June), we haven't had any thunderstorms yet. I prefer not to write about it till I know whether it all works.

The second project is a working greenhouse. Since the solarium was a control bust (even though the solarium/wood stove combination appears to have reduced the house heating oil requirement from 1350 gallons to about 550 for last winter), I stated at the end of that article that a real greenhouse must surely benefit from control. Well, I built an 11' x 20' glass/redwood greenhouse and the jury is still out. I installed solar-controlled vents and a thermostatically controlled vent fan. With the trees providing some shade during the day, the temperature appears fairly manageable.

Automatic watering seemed the most obvious control application, but I am still experimenting. Only owners of greenhouses know that about the best way to water the plants is to stand in the doorway with a **firehose** on wide spray. Of course, regulated drip systems and intermittent watering might be better for certain plants. At this point I'm still collecting data.

So, I'm not stalling, Curt. I'm just trying to justify hanging a \$200 water spray controller over a plant that would be just as happy sitting in once-a-week water-filled tray. Of course, all of this is contingent on the two lightning rods on the greenhouse roof keeping said controller from being incinerated anyway.

