CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

PROGRAMMABLE DEVICES

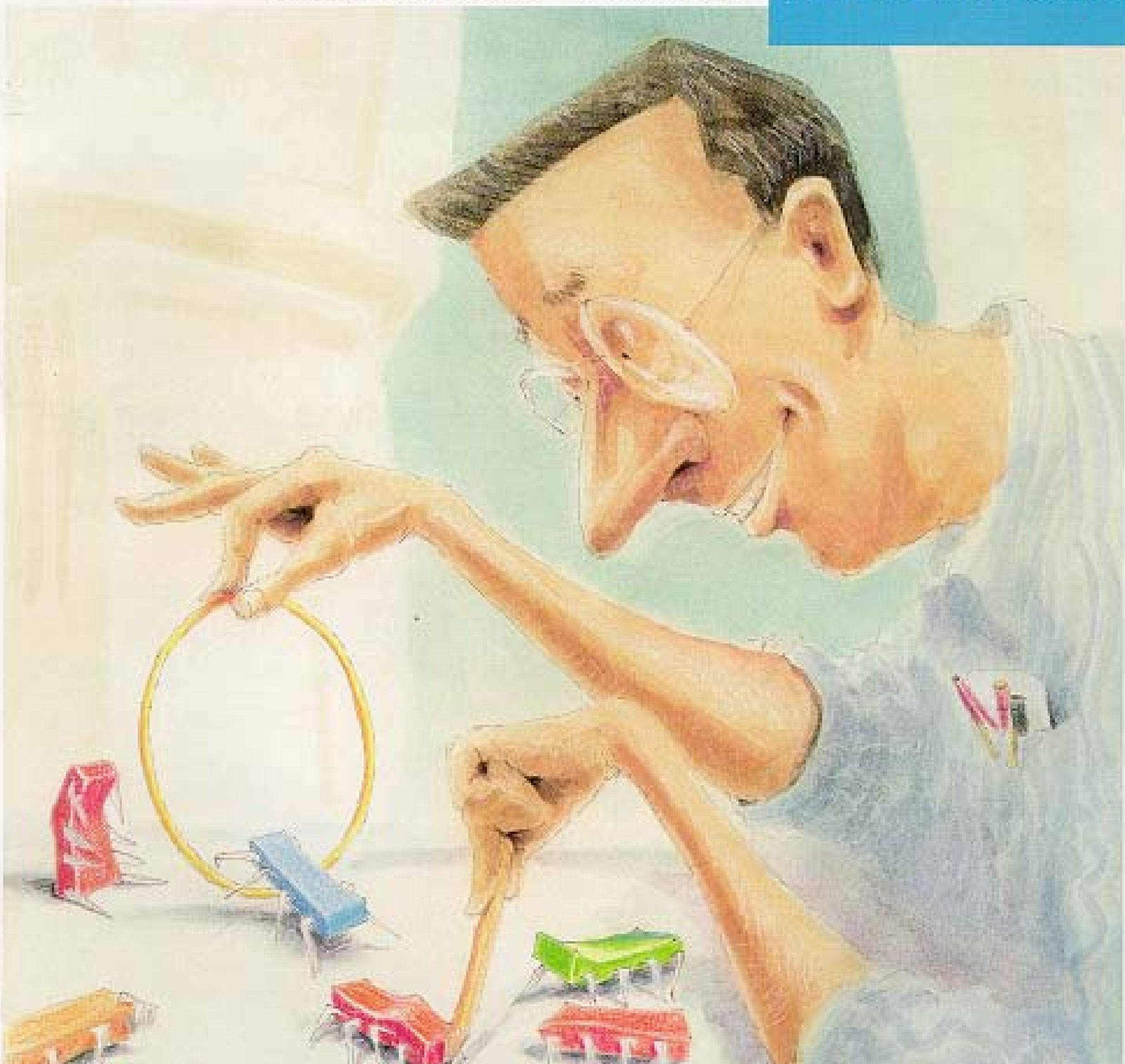
Interleaved Bursting EPROM Interface for the M68040

Introduction to FPGAs

Reading Credit Card Magnetic Stripes

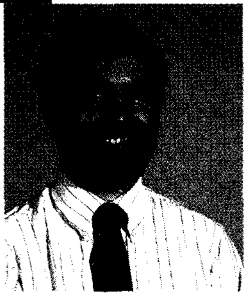
PC BIOS Extensions in C

November 1993 — Issue #40



$3.95 U.S.
$4.95 Canada

EDITOR'S INK



Circuit Cellar and the Internet

Circuit Cellar and the Computer *Applications Journal* have always been at the forefront of technology, not afraid to try new equipment, ideas, or techniques. When we first started putting together the magazine, we entered the world of desktop publishing when it was still in its infancy. We were one of the first regular publications to also run a BBS, making article-related code available to download and providing a forum in which readers, authors, and editorial staff could freely exchange ideas. Now, we at Circuit Cellar make it easier for people from all over the world to contact us at little or no expense: via the Internet.

Before you get too excited, I'm not talking about logging onto the Circuit Cellar BBS using telnet or ftp. Right now, making the BBS available through these services is just too expensive. Rather, the BBS now has full Internet Email access.

Too busy to write up a formal letter to the editor, address an envelope, and put it in a mailbox? Quickly jot down your ideas and Email them to us. Do long-distance charges (especially from overseas) make you break out in a sweat just thinking about them? If you have an Internet account, you can send us mail or request article-related files without any phone calls. If you've always wanted your own Internet address and mailbox, but didn't want to pay a commercial service for them, you automatically get them at no cost when you call and use the Circuit Cellar BBS by modem.

We're still in the early stages, so things may need ironing out, but we're very excited about this new service. To get more details, call the Circuit Cellar BBS with your modem (203/871-1988, 8N1) and download the information file, or send Email to ftpmail@circellar.com and include the phrase "get help.txt" in the body of the message. The file will be mailed back to you automatically.

Briefly summarizing our editorial offerings this month, we start out with a design idea intended to speed up M68040 designs that use EPROMs. Rather than using expensive, single-source parts, we show you how to use standard, off-the shelf EPROMs in an interleaved bursting configuration.

Next, part one of a two-part article gives an introduction to field-programmable gate arrays, what is available on the market, and how to decide what to use. Part two will give a design example.

Our third feature presents a low-cost alternative to traditional EPROM emulators that requires little or no extra hardware.

In our columns, Ed extends his discussion of BIOS extensions by writing one in Micro C; Jeff uses DTMF over the phone in a way its original designers probably never intended; Tom expands on last month's all-in-one chips with even more powerful versions; John takes a look at decoding credit card magnetic stripes; and Russ picks out patents related to our theme: Programmable Devices.

Ken

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

EDITOR-IN-CHIEF
Ken Davidson

TECHNICAL EDITOR
Michael Swartzendruber

ASSOCIATE EDITOR
Rob Rojas

ENGINEERING STAFF
Jeff Bachiochi & Ed Nisley

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
John Dybowski & Russ Reiss

NEW PRODUCTS EDITOR
Harv Weiner

ART DIRECTOR
Lisa Ferry

GRAPHIC ARTIST
Joseph Quinlan

CONTRIBUTORS:
Jon Elson
Tim McDonough
Frank Kuechmann
Pellervo Kaskinen

Cover Illustration by Bob Schuchman
PRINTED IN THE UNITED STATES

PUBLISHER
Daniel Rodrigues

PUBLISHER'S ASSISTANT
Susan McGill

CIRCULATION COORDINATOR
Rose Mansella

CIRCULATION ASSISTANT
Barbara Maleski

CIRCULATION CONSULTANT
Gregory Spitzfaden

BUSINESS MANAGER
Jeannette Walters

ADVERTISING COORDINATOR
Dan Gorsky

CIRCUIT CELLAR INK, THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (203) 675-2751. Second class postagepaidatVernon, CT and additional offices. One-year (12 ISSUES) subscription rate U.S.A. and possessions$21.95. Canada/Mexlco$31.95,allothercountries $49 95. All subscription orders payable i" U.S. funds only. via international postal money order or check drawn on US bank. Direct subscription orders and subscription related questions to The Computer Applications Journal Subscriptions, P 0. Box 7694. Riverton, NJ 06077 or call (609) 786-0409.
POSTMASTER Please send address changes to The Computer Applications Journal, Circulation Dept., P.O. Box 7694, Riverion. NJ 08077.

HAJAR ASSOCIATES NATIONAL ADVERTISING REPRESENTATIVES

NORTHEAST
Debra Andersen
(617) 769-8950
Fax: (617) 769-8982

MID-ATLANTIC
Barbara Best
(908) 741-7744
Fax: (908) 741-6823

SOUTHEAST
Christa Collins
(305) 966-3939
Fax: (305) 985-8457

MIDWEST
Nanette Traetow
(708) 789-3080
Fax: (708) 789-3082

WEST COAST
Barbara Jones
& Shelley Rainey
(714) 540-3554
Fax: (714) 540-7103

CircuitCellarBBS-24 Hrs 300112001240019600114 4k bps.6 bits, noparity, 1 slop bit,(203)871-1988;2400/9600 bps Courier HST, (203) 871-0549

INSIDE ISSUE 40

14 **Speed up Your M68040 with an Interleaved Bursting EPROM Interface**
by Ron Stence

26 **Designing with FPGAs/Part 1:** An Overview
by Del Hatch

36 **Loader31: A Pseudo EPROM Emulator**
by Brad Hunting

42 ❑ **Firmware Furnace**
Booting C/Writing a Micro-C BIOS Extension for the '386SX Project
Ed Nisley

56 ❑ **From the Bench**
Talk on the Phone Without a Word
Jeff ***Bachiochi***

62 ❑ **Silicon Update**
The Swiss Army Sends in the Special Forces
Tom Cantrell

68 ❑ **Embedded Techniques**
Decoding Magnetics
John Dybowski

2 Editor's INK
Ken Davidson
Circuit Cellar and the Internet

6 Reader's INK
Letters to the Editor

8 New Product News
edited by Harv Weiner

78 Patent Talk
Russ Reiss

84 ConnecTime
Excerpts from the Circuit Cellar BBS
conducted by Ken Davidson

96 Steve's Own INK
Steve Ciarcia
Lest We Forget that Chips Can...and Will

81 Advertiser's Index

READER'S INK

DON'T TRY THIS AT HOME

The whimsy of Bob Schuchman's artwork on the magazine's covers is usually enjoyable and often evokes a smile. However, the August cover is not one of these. I suspect it did not occur to you what message could be sent to youngsters or others unfamiliar with the power of electricity by this picture. Would they stick their fingers in light sockets? Maybe not, but it is amazing what young people can do once it is suggested to them.

I would suggest that you look at the covers in light of all possible audiences-the unintended observers who could drastically misinterpret what is shown as well as the professionals.

H. Dick Breidenbach
W. Bloomfield, Mich.

A SATISFIED CUSTOMER

I recently picked up a copy of the ***Computer Applications /ournal,*** issue #38, September 1993, and wanted to make some comments.

I have been working with electronics since 1959. My interest in that field started as a young man interested in Amateur Radio. I still hold an advanced class Amateur Radio license but found that operating a radio station was not nearly as rewarding as building stuff. I have been building "stuff" ever since.

Computers were a natural extension of my original interest in electronics, and I got my first taste of their potential in the late 1970s. I built an 1802-based, tape programmable microcomputer that used tiny BASIC. Later, I got an S-100 system running CP/M 2.2. Since then I have continued to progress through the various Intel processors.

My involvement with personal computers reads like a history of that emerging industry over the last 15 years. My interest has not waned. If anything, it has intensified and become focused toward applications. Hands-on hardware projects have always been my first love; my second love being the programming to make the hardware work in custom applications.

I am, and always have been, an avid reader. The publications I find most enjoyable are those that deal with hands-on-project-oriented topics. In the '7Os, Wayne Green's 73 magazine was hard to beat. Then came ***BYTE,*** which was really superb in its earlier days. It reached its zenith when "Ciarcia's Circuit Cellar" was most active. It was information from ***BYTE*** that allowed me to add a hard disk drive to my first S-100 system.

I am the sort of person who reads ***Scientific American*** for the "Amateur Scientist" column. Much the same way I used to read ***BYTE*** for Steve's column. When ***BYTE*** became more consumer oriented with reviews and ads replacing its earlier construction articles, my only reason for reading it was Steve's column. When the column disappeared, I stopped regarding ***BYTE*** as a serious publication.

I am very pleasantly surprised to see that Circuit Cellar has started its own magazine. After reading the complimentary issue you sent, I have decided to subscribe. I feel that your magazine is destined to become the advanced hardware hacker's/engineer's equivalent of ***Dr. Dobb's /ournal.***

I am also interested in being a contributor to your magazine. I find that there is not a shortage of applications, just a shortage of appliers. Ideas for applications abound. Being an engineer with considerable experience in a variety of industries puts me in a position where I can draw on my experience for applications ideas.

I look forward to working with you and hope to make significant contributions to our mutual benefit. I also look forward to receiving my next issue of the ***Computer Applications Journal.***

Frank Kamp
Richardson, Tex.

COLLEGE COMMENTS CONTINUE

In Steve's editorial, "The Collegiate Challenge" in the September 1993 issue, he suggests that the ". . .student who faces the same material as a personal challenge, trying to learn all he can; trying to absorb as much as he can; trying to rise to the best of his abilities" may be the better student. I thought so too, but...

I fell into that trap and did not realize I was wasting my energy. It was difficult for me to pass the exams because I could not memorize well.

By applying my engineering skills to the education process, "the piece of paper," I found literature that suggested that people remember in a combination of three ways: visual, verbal, and kinesthetic. I then found ways to enhance the kinesthetic learning process. Applying engineering methodology and medication, 15 years later I graduated with a 4.0 in an associate's degree program and then an engineering program. Before that, I graduated with a 3.8 in my senior year of high school and had a 1.6 GPA after two years in college in an electrical engineering program.

READER'S INK

That 1.6 GPA, I believe, was primarily a result of using too much energy to understand the concepts. I did not realize at this point that I had to pass the exam without tools. I had to memorize the tools first. I could not derive the tools because of time constraints during the exam. I wasted energy.

Engineering is a methodology, not a course in what you can memorize. The stupid education process of "memorizing, examining without your tools (books, notes, etc.], and promptly forgetting" has got to stop. It makes you wonder why some of the most successful people (entrepreneurs) didn't go to college. Are they the smart ones?

I spent the summer working with a senior engineering student. I had given him material to use as a guide. It was interesting to watch him apply it verbatim to a problem. It would not work. Is this the engineer of the future? One of his greatest assets was realizing that after a 4-year engineering degree, he doesn't know anything.

Ron Dozier
Wilmington, Del.

Contacting Circuit Cellar

We at the Computer Applications *Journal* encourage communication between our readers and our staff, so have made every effort to make contacting us easy. We prefer electronic communications, but feel free to use any of the following methods:

Mail: Letters to the editor may be sent to Editor, The Computer Applications Journal, 4 Park St., Vernon, CT 06066.

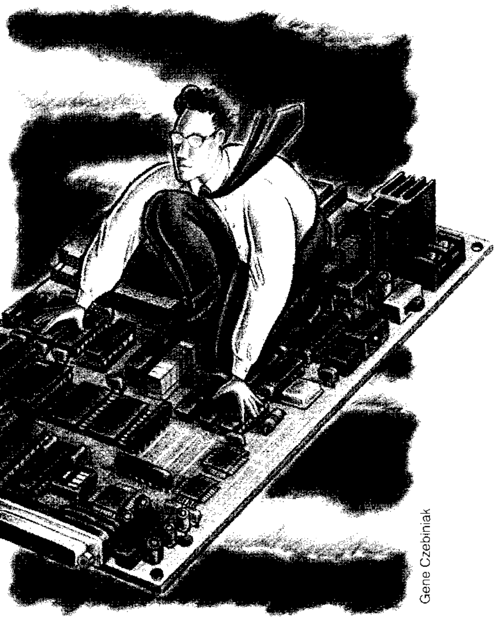
Phone: Direct all subscription inquiries to (609) 786-0409. Contact our editorial offices at (203) 8752199.

Fax: All faxes may be sent to (203) 872-2204.

BBS: All of our editors and regular authors frequent the Circuit Cellar BBS and are available to answer questions. Call (203) 871-1988 with your modem (300-14.4k bps, 8NI).

Internet: Electronic mail may also be sent to our authors and editors via the Internet. To determine a particular person's Internet address, use their name as it appears in the masthead or by-line, insert a period between their first and last names, and append "@circellar.com" to the end. For example, to send Internet Email to Jeff Bachiochi, address it to jeff.bachiochi@circellar.com.

No bugs on board.



Of course, what else would you expect from the acknowledged leader in Intel 80C186 and NEC V-Series embedded system software development tools.

Start with the complete Borland C++ and Microsoft C/C++ application templates from Paradigm LOCATE. Add your code and check it out in your target system with the highly acclaimed Paradigm DEBUG, either stand-alone or with popular in-circuit emulators from Intel, NEC, Applied Microsystems, CheckMate, Microtek, ZAX, and others.

If you get stuck, call our toll-free technical support hotline for free technical support. Then rest assured that your embedded application is rock-solid and free from those embedded system nasties.

After all, life is short and you can't play hard when you're stuck in the lab fixing bugs. Get Paradigm, get bug-free, and get on with your life.

PARADIGM

Proven Solutions for Embedded C/C++ Developers

1-800-537-5043

Supports Borland® Tools™

Paradigm Systems ▪ 3301 Country Club Road, Suite 2214 ▪ Endwell, NY 13760 ▪ (607) 748-5966 ▪ FAX: (607) 748-5968

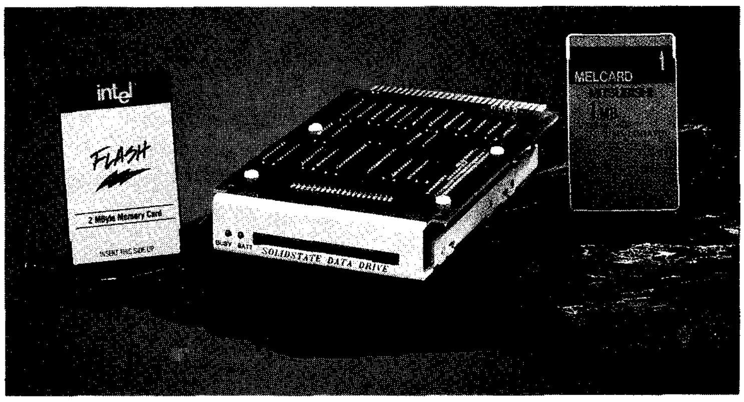
All trademarks are property of their respective holders.

#103

NEW PRODUCT NEWS

Edited by Harv Weiner

"CREDIT CARD" STORAGE SYSTEM



WinSystems has introduced an alternative to the floppy disk. The MCM-RSSD-J Removable Solid-state Disk Drive is designed to replace conventional rotating disk memories in harsh applications where computers are subject to extended temperature, magnetic fields, vibration, dust, dirt, fumes, and so forth. The unit is totally devoid of spindle and stepper motors, disks and bearings-components that can fail under severe operating conditions. It is designed to store programs and data for applications such as data collection and logging, loading program updates, diagnostics, portable instruments, and in industrial environments where floppy disks cannot survive.

The MCM-RSSD-J is ideal for STD bus-based embedded systems and is compliant with the JEIDA memory card standard. It supports the 68-pin PCMCIA data cartridges for memory storage of up to 64 megabytes. The IC memory card is small, light, and durable. The drive will read SRAM, OTPROM, EEPROM and flash memory cartridges, and will write SRAM and flash memory cards.

The drive mounts directly on an STD bus card to form a complete storage subsystem which plugs directly into the STD bus card cage. The drive is equipped with panel status LEDs which include Low Battery and Busy. The Low Battery LED shows the status of the battery and indicates when it needs to be replaced. The Busy LED indicates when the card buffers are enabled and that it is unsafe to insert an IC memory cartridge.

The MCM-RSSD-J appears as an IDE interface to a host STD bus microcomputer. An installable device driver that supports both the DOS and ROM-DOS operating systems is available, and the application software then treats the MCM-RSSD as a disk drive. An EPROM socket is on board to permit an optional BIOS extension to enable STD bus XT- or AT-compatible systems to boot from the MCM-RSSD-J. The MCM-RSSD-J sells for $395.00. A CMOS STD bus version, the LPM-RSSD-J, sells for $425.00.

WinSystems, Inc.
715 Stadium Dr., Ste. 100 • Arlington, TX 76011-6225
(817) 274-7553 • Fax: (817) 548-1358

#500

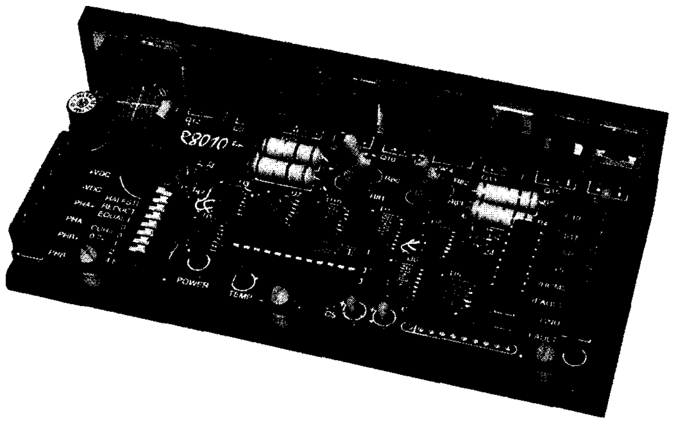
STEPPER MOTOR DRIVER

A new, high-power (lo-amp) stepper motor driver has been announced by MicroKinetics Corporation. The DR8010 features full- and half-stepping modes, equalized torque half-stepping, DIP-switch-selectable current, automatic idle current cutback, as well as full antishorting and thermal protection. The use of multilayer printed circuit board fabrication and surface mount technology results in a very compact and reliable design. Overall measurements are 6"x2.5"x1.5".

Unique to this design is the equalized torque half-stepping, which smoothes out the uneven power difference between adjacent steps in typical half-stepping drives. The DR8010 is suitable for many CAD/CAM, machining and manufacturing applications. The DR8010 sells for $345.00.

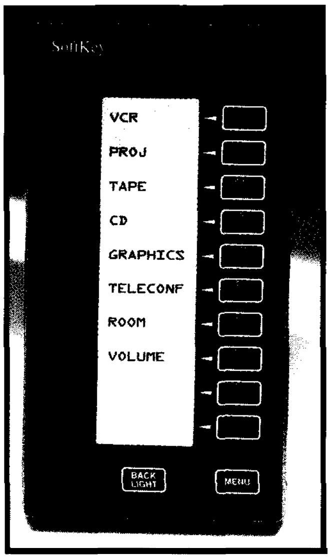
MicroKinetics Corp.
1220-J Kennestone Cir. • Marietta, GA 30066
(404) 422-7845 • Fax: (404) 422-7854

#501



NEW PRODUCT NEWS

WIRELESS PROGRAMMABLE TRANSMITTER



The TX-SK+SoftKey wireless transmitter from AMX Corporation represents the latest advance in programmable, menu-driven wireless remote control. The SoftKey's on-board microprocessor, multiple control screens, and ten function keys offer a wide array of features.

All transmitters include radio frequency (RF) and infrared (IR) capabilities and can also transmit IR codes from other manufacturers. This feature allows the SoftKey to act as a mini control system, directly controlling VCRs, audio components, and video projectors.

The hand-held SoftKey transmitter weighs approximately 17 ounces and features an illuminated vertical liquid crystal display (LCD) that can display up to 30 text lines.

A rechargeable NiCd battery pack is a standard feature capable of approximately 14 hours of operation with an S-hour charge. Low battery and charging indicator messages flash across the bottom of the display screen.

All aspects of SoftKey operation, including menus, text, operation logic, and wireless code transmission are programmable using AMX SKDESIGN graphical or AXCESS software. The SoftKey has 24K of permanent user program memory. Both compiled and source code can be stored for most programs.

SoftKey can be programmed to provide bidirectional RS-232 control commands via the unit's Master/Control port. The RS-232 port can be configured to operate as an AXCESS Master port for program uploading and downloading, or as an RS-232 Control port. The TX-SK+SoftKey suggested retail price is $1590.00.

AMX Corp. ● 11995 Forestgate Dr. • Dallas, TX 75243
(214) 644-3048 • Fax: (214) 907-2053

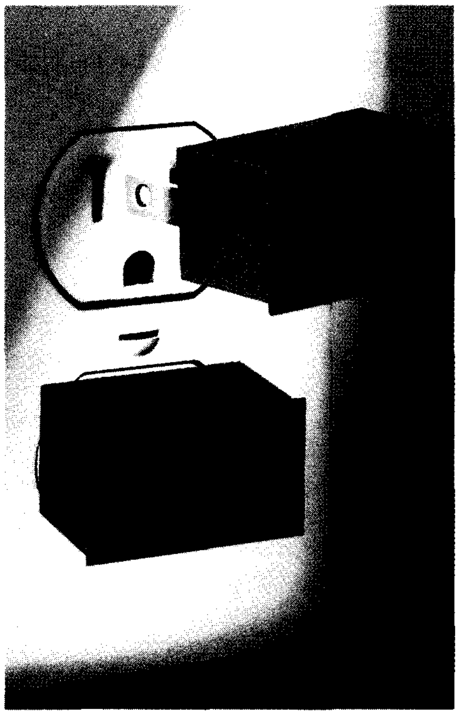
#502

DIGITAL AC LINE MONITOR

DMS-20PC-l-LM is a component-sized, self-contained, AC voltmeter for true stand-alone measurements. It requires no additional components or auxiliary power. Simply plug it into any wall outlet and instantly read line voltages from 85 to 264 VAC (47-63 Hz). The large (0.37") bright-red LED display makes the unit easily readable in virtually any lighting condition.

The DMS20PC- 1 -LM employs half-wave sinusoidal averaging techniques (RMS calibrated) and has a sampling rate of 2.5 samples per second. It features a display resolution of ±1 VAC over the full input range of the meter. The unit measures 1.38"xl .OO"xO.88" and is packaged in a red filter case with integral bezel. Maximum power consumption is 0.5 W. The unit is fully encapsulated for ruggedness. A low parts count and surface mount technology assure reliable trouble-free operation. All units are overvoltage protected to 300 VAC, and the operating temperature range spans -25°C to +60°C.

The DMS-20PC- 1 -LM can be used for industrial, laboratory, office, and field service applications. Its miniature size is perfect for design into high-end consumer electronics, laboratory instrumentation, and other products requiring accurate AC line monitoring.



The DMS20PC- 1 -LM AC Line Monitor sells for $45.00 and is available with screw terminals for panel mounting and blade or round terminals for socket insertion.

Datel, Inc. • 11 Cabot Blvd. • Mansfield, MA 02048
(508) 339-3000
Fax: (508) 339-6356

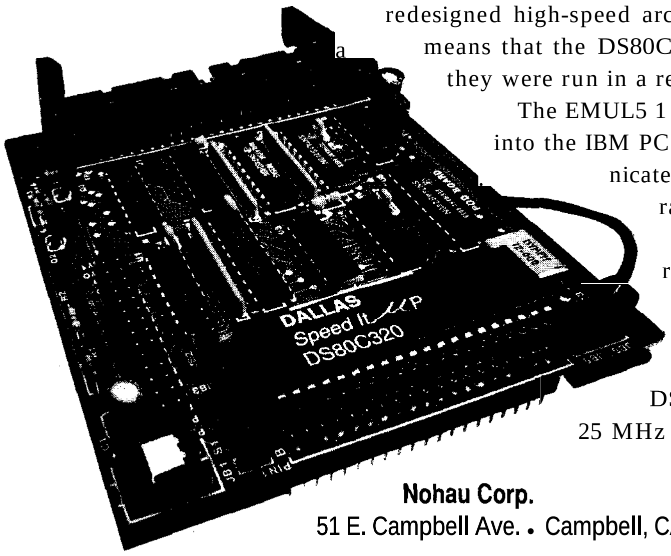
#503

NEW PRODUCT NEWS

IN-CIRCUIT EMULATOR

The **EMUL51-PC** from Nohau is a high-performance in-circuit emulator specifically designed to give an optimized environment to develop 8051-family hardware and software. A special device from Dallas Semiconductor enables Nohau to achieve emulation capabilities for the DS80C320.

The DS8OC320 is pin compatible with the 8OC32 and uses the standard 805 1 instruction set. The core is a redesigned high-speed architecture which removes the 8051's "dead cycles." This means that the DS80C320 will run instructions 1.5 to 3 times faster than if they were run in a regular 8OC51 using the same clock frequency.



The EMUL5 1 -PC emulator consists of a board which plugs directly into the IBM PC/XT/AT bus. It is also available in a box which communicates with the PC through a standard RS-232 channel at data rates up to 115k bps.

An optional trace board holds up to 256K trace records of 64 bits each. It has multiple trigger levels, filtering, loop counting, on-the-fly timing, and time stamp.

The EMUL51-PC/DS320 sells for $3595.00. The DS80C320 probe, POD-C320 (shown), runs at speeds up to 25 MHz and sells for $995.00.

Nohau Corp.
51 E. Campbell Ave. • Campbell, CA 95008 • (408) 866-1820 • Fax: (408) 378-7869

#504

SPECTRUM ANALYZER IN A PROBE

A low-cost scope accessory probe that can be used for EM1 investigation is available from Smith Design. The 255 **Spectrum Probe** can be used to analyze undesired signals radiated or conducted by equipment and to view incoming signals that could cause interference.

The probe covers a frequency range from 30 kHz to 2.5 MHz and features a 60-dB [min.) dynamic range. It features a tangential sensitivity of 40 µV ±3 dB at 1 MHz with a flatness of ±2 dB from 0.05 to 2.5 MHz. Its input loading is 10 pF and tip conducted radiation is nil. The internally shielded 255 Spectrum Probe is 7.5" long, 0.9" in diameter, and weighs only 2 oz.

The 255 Spectrum Probe can be used to measure RF voltage or current entering or exiting electronic equipment; estimate and reduce the field strength of emissions; check the HF portion of infrared, ultrasonic, and line-conducted remote controls; sniff for sources of EMI; and indicate the source and path of RF voltages and currents on PC boards. The conducted limits for FCC Parts 15 and 18 and VDE 0871 fall mainly within the range of the probe, so FCC compliance can be measured.

The 255 Spectrum Probe sells for $279. A model 107 Spectrum Probe is available to cover the 1 -MHz to 100-MHz range and sells for $249 ($279 internally shielded).

Smith Design
207 E. Prospect Ave. • N. Wales, PA 19454
Phone/fax: (215) 661-9107

#505

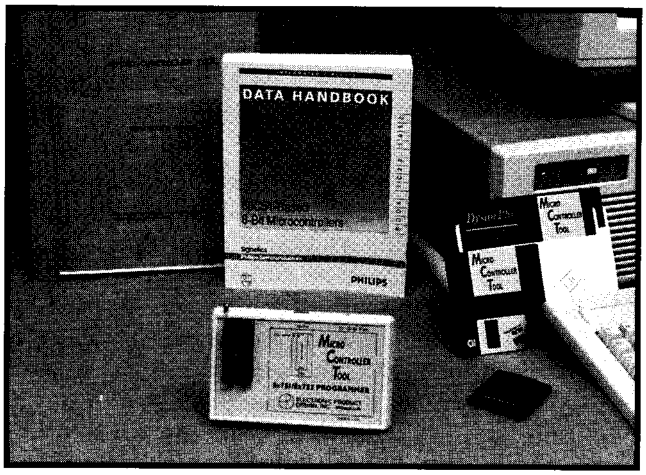
NEW PRODUCT NEWS

PC-BASED DEVELOPMENT SYSTEM

The MicroController Tool (MCT) is a complete, low-cost, PC-based development system for the Signetics 87C751 and 87C752 single-chip microcontrollers. As an optimized, integrated, and menu-driven package, MCT includes a project manager, text editor, assembler, and programmer. The global integration of MCT software and features of the 87C751/752 enable the engineer and small business to quickly develop and produce low-cost microcontroller-based products.

The high-performance 87C751 features 64 bytes RAM, 2K EPROM, and 19 I/O ports. The 87C752 offers two additional I/O ports and a 5-channel, 8-bit ADC. Both microcontrollers are derivatives of the 805 1 family and share the same instruction set.

The MCT development system sells for $399.00 and comes complete with a Microcontroller Handbook, Operator's Manual, software, and a Serial Programming Module with AC adapter. MCT accessories including prototyping kits are also available.



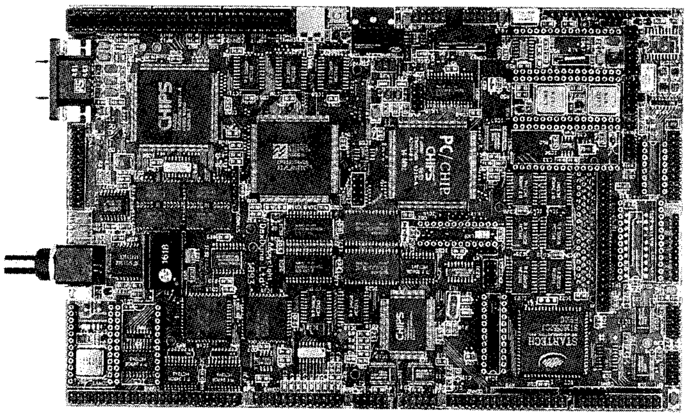
Electronic Product Design, Inc.. 6963 Bluebelle Way
Springfield, OR 97478 . (503) 741-0778 #506

TotalPc™ - *the* embedded system

High integration compact PC controller.

STANDARD FEATURES OF *TotalPc*™

- 14 MHz PC CPU with CGA interface for LCD/CRT display
- Standard PC keyboard I/F and 12 x 8 matrix keypad interface
- Memory space for up to 2M byte EPROM, 4M byte of RAM and 4MBytes of FLASH
- Watchdog timer, Time of Day clock
- Floppy, IDE, Printer and 5 serial RS232 interfaces, two with optional isolated RS485
- Thin Ethernet I/F
- VGA support for LCD and CRT
- PCMCIA-2 I/F with hot insert capability
- 24 Opto 22 compatible buffered industrial parallel I/O lines
- 8 single or 4 differential input 12 bit ADC and 8 output 8 bit DAC
- Fully buffered PC Expansion slot
- Single 5V operation. Low power mode for battery operation



If you are making POS terminals, GPS systems, field portable instruments, factory data terminals, Ethernet based outstations or plain process control systems for industrial environments you will find that ***TotalPc***™ ***gets*** to the heart of your problem. Our EPROM based Version 5.0 DOS provides a known and proven development environment for your application software on the ***TotalPc***™

OEM and customisation enquiries welcomed.

dex dyne

DEXDYNE LIMITED

15 Market Place
Cirencester
Gloucestershire **GL7 2PB**
England

Tel: 0285 658122
Fax: 0285 655644

#1 04

NEW PRODUCT NEWS

IDE BIOS FOR EMBEDDED SYSTEMS

An **Embedded BIOS IDE Development Kit** will enable embedded system developers to manufacture their own IDE-compatible ROM BIOS and customize it to meet the special needs of their embedded hardware has been announced by General Software. Embedded BIOS IDE comes with full source code (over 30,000 lines of well-structured assembly language], over 112 configuration options, a complete set of ROM building utilities, a ROM disk BIOS extension module, remote disk software, and General Software's BIOS-aware debugger.

The new BIOS includes a setup screen that can be routed to a PC screen or to a serial port using ANSI escape sequences. Default setups are included for drive types when no setup is possible. The primary focus of Embedded BIOS IDE is the support of AT-compatible BIOS functions in a real-time environment, with a low interrupt latency of less than 10 instructions.

Embedded BIOS IDE costs $1145.00. Royalties are $1 .OO/copy, and a free product demo disk is available.

General Software, Inc. • P.O. Box 2571 • Redmond, WA 98073 • (206) 391-4285 • Fax: (206) 557-0736 #507

MATH ACCELERATOR AND EMULATOR

QuickWare has announced the release of Q387 **Version** 3.5 **Math Accelerator and Emulator. Q387** is a cost-effective alternative to an 80x87 math coprocessor. The program allows all owners of 486SX-, 486SLC-, 386DX-, and 386SX-based PCs to run programs which normally require a math coprocessor. Q387 is very fast, speeding up coprocessor-optional graphics and CAD programs by as much as 400%. Q387 is compatible with all DOS, DOS extender, and Windows applications.

For laptop and notebook owners, Q387 extends battery life by up to 25%. The program draws power only when math operations are performed, as opposed to a coprocessor, which drains battery power at all times.

Q387 is distributed as an upgradable demo and may be obtained free of charge by modem from CompuServe in the IBMHW forum, America OnLine, the QuickWare BBS at (512) 292-1212, the QuickWare Demo Line at (512) 280-6707, or many local BBSs. Q387 registration is $25.00 and may be accomplished toll free by phone.

QuickWare • P.O. Box 684652 • Austin, TX 78658 (512) 280-1452 • 70750.2147@compuserve.com #508

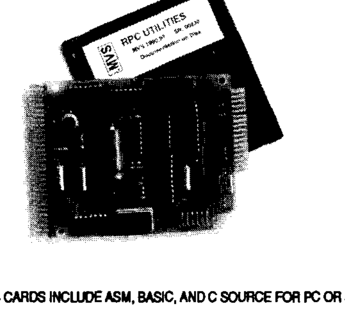
REMOTE POWER CARD!

3 VERSIONS:

RESET
WATCHDOG RESETS PC IF IT HANGS FOR HARDWARE OR SOFTWARE REASONS

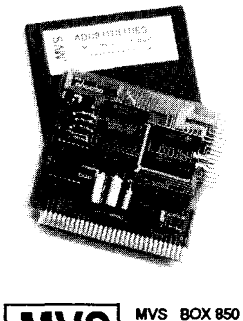
PHONE
TURN ON PC WITH PHONE, SHARE VOICE / MODEM LINE, CONTROL AC APPLIANCES

TIMER
WAKEUP OR SHUTDOWN PC, LATE NITE BACKUP / MODEM, CONTROL AC APPLIANCES



**95$** 27$ OEM

ALL MVS CARDS INCLUDE ASM, BASIC, AND C SOURCE FOR PC OR SBC



8 CHAN ADC
DATA AQUISITION, SERVO CTL, AUDIO
8-BIT RESOLUTION 22KHZ SAMPLE RATE
SHARP CUTOFF ANTI-ALIAS FILTER
CREATE STEREOBLASTER(.VOC) FILES
95$

2 CHAN DAC
VOICE MAIL, MUSIC, ALARMS, CTL VOLT
8-BIT RESOLUTION 44KHZ SAMPLE RATE
PLAYS MONO / STEREO BLASTER FILES
FUNCTIONS AS DIGITAL ATTENUATOR TOO
75$

MVS
MVS BOX 850
MERRIMACK NH
(508) 792 9507

5 YEAR LIMITED WARRANTY
FREE SHIPPING IN USA

#105

µLAN

The 9-Bit Solution is the Answer to your Embedded Network Needs

PC/XT/AT
8051
8096
80C186EB/EC
68HC11
68HC16
Z180

The Cimetrics Technology 9-Bit Solution is a complete microcontroller network (µLAN) that supports the 8051, 68HC11, 80186, and many other popular processors. The 9-Bit Solution takes full advantage of multiprocessor modes built into microcontroller serial ports. Our flexible software and hardware allow developers to create powerful, yet inexpensive master/slave multidrop embedded controller networks.

- Up to 250 nodes
- 16-bit CRC error checking with sequence numbers
- Low network overhead and low resource requirements
- RS-485 interface card for the PC
- Complete source code included
- Comprehensive documentation

CIMETRICS TECHNOLOGY
120 West State Street, Ithaca, NY 14850
TEL: (607) 273-5715 FAX: (607) 273-5712

#106

NEW PRODUCT NEWS

DUAL-BATTERY FAST CHARGE IC

The first integrated solution for fast charging one or two nickel metal hydride (NiMH) or nickel cadmium (NiCd) batteries is available from Benchmarq. The CMOS **bq2005 Dual-battery Fast Charge IC** provides comprehensive fast charge control functions with high-speed switching power-control circuitry for one or two independent battery-pack systems.

The bq2005 is the basis of a cost-effective solution for sequentially charging two battery packs using flexible control of constant-current or current-limited charging supply. The 20-pin, 300-mil bq2005 PDIP or SOIC package can be used as a frequency-modulated controller operating up to 300 kHz for switched regulation of the charging current. The bq2005 may alternatively be used with a linear regulator or transistor to gate an external supply.

The bq2005 supports both NiCd and NiMH batteries, allowing upgrades from designs for NiCd to new NiMH battery technology without redesign. The chip supports charge rates of 1 to 1.5 hours for NiMH and as fast as 10 to 20 minutes for appropriate NiCd batteries.

Fast charge begins with the application of the charging supply or by replacement of the battery. For safety, charge is inhibited until the battery temperature and voltage are within configured limits. Temperature, voltage, and time are monitored throughout fast charge. The bq2005 includes a "top-off" charge at 12.5% of the charge rate for NiMH batteries, as well as a user-selectable pulsed trickle charge of $C/32$ or $C/64$.

The bq2005 incorporates temperature slope ($\Delta T/\Delta t$) sensing for fast charge termination. It calculates the slope of the battery temperature rise curve and uses the rapid temperature increase associated with fully charged batteries to terminate charging quickly when the rate of temperature increase is outside predetermined acceptable limits. In addition to $\Delta T/\Delta t$, the bq2005 terminates charge based on a sensitive negative delta voltage detection. Fail-safe terminations include maximum temperature, maximum charge time, and maximum battery voltage. The bq2005 sells for $4.70 (PDIP) and $4.80 (SOIC) in I k quantities.

Benchmarq Microelectronics, Inc.
2611 Westgrove Dr., Ste. 109 . Carrollton, TX 75006
(214) 407-0011 . Fax: (214) 407-9845 #509

▶ Project Part5 ◀

New Stuff	
TDA8444 I^2C bus octal 6-bit DAC	9.30
MT8888 DTMF Transceiver (Intel bus interface)	10.50
PCF8583 I^2C bus clock/calendar/counter w/ 256 byte RAM	11.70
UDN2998W Dual full-bridge stepper driver 50 V, 2 A	13.80
PCF8591 I^2C bus quad 8-bit ADC. single 8-bit DAC	15.10
BQ2003 Fast-charge NiCd/NiMH battery controller	14.55
MT8808 8x8 bipolar signal analog crosspoint	16.10
BQ2001 Battery Energy Management Unit	22.75
HBCS-1100 HP high-resolution barcode sensor	32.80
CH1840 FCC pre-approved DAA (full function, street legal phone hookup)	72.50
Firmware Furnace Embedded **'386SX Stuff**	
Complete schematics (15 pages or so)	5 00
DS2400 Silicon Serial Number (2 chips)	8 40
MAX691 Power supervisor	11 85
Firmware Development Board ICs 8 parts (through Issue 36)	45.70
Firmware **Flyers** (prices are postpaid in US / with any pans order)	
8051 Firmware Debugging Techniques (65 pages w/ 3 5" diskette)	18 /15.00
Introduction to the 8051 Instruction Set (46 pages)	10 / 8.00
8051 Instruction Reference Card	3 / 2.00
8255 Cheat Sheet Reference Card	3 / 2.00
IR Remote Control	
LD273 dual IR LED (bright, wide beam)	2.10
IR3C02A laser diode controller (±5, for LT022MC/LN9705P laser)	2.30
IR3C07 laser diode controller (+5V, for LT022PD/LN9705 laser)	2.85
PH302 fast IR photodiode	3.10
GP1U52Y 40 kHz IR receiver (side-looking)	4.00
IS1U60 38 kHz IR receiver	4.80
IRSAMPLE parts (PH302, LM311, etc See MCIR-Link article INK 29)	5.70
Excellent IR filter (opaque to visible light, 35 mm slide mount)	6.75
MC145030 IR encoder/decoder	6.75
IR I/O: IS1U60, LD273, CD4047, 2N2907, red LED. schematic (IR-Link!)	10.40
Stepper Motor 8 **Power Drivers**	
ULN3751Z Power op amp (±3V to ±13V supply. 3 5 A output)	4.60
UDN2993B Dual full H-bridge bipolar stepper motor driver	5.45
UCN5841A Serial-input, 8 latched 500 mA sink drivers	6.90

. ..and much more...

IPS Ground/2nd day $6 519 to 46 US slates, COD add $4 50. PO Boxes and Canadian orders $6 for USPS mail. Check, MO, or COD only, no credit cards. POs add $50. call first NC residents add 6% sales tax Quantity discounts start at five parts. Data sheets included with all parts.

Call/write/fax for seriously tempting catalog...

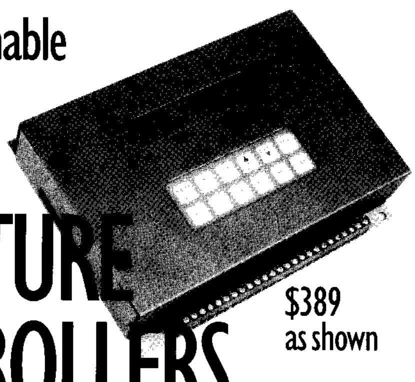
Pure Unobtainium

▶ Your unusual parts source ◀

13109 Old Creedmoor Road . Raleigh. NC 27613

FAX/voice (919) 676-4525

C-Programmable
Low Cost
Expandable
MINIATURE
CONTROLLERS



$389
as shown

Use our miniature controllers in your next product. in your plant or in your test department. Get a wide range of analog and digital inputs and outputs, relay and solenoid drivers, DACs, clocks and timers. All our devices provide serial communications (RS232 and RS485) and battery-backed RAM. Options include LCDs up to 8 x 40. keypads, and expansion cards. Our easy-to-use. yet powerful, Dynamic C ™ development system is only $195. Prices start at $159, quantity one.

Z WORLD
ENGINEERING

1724 Picasso Ave.
Davis, CA 95616
916.757.3737
916.753.5141 FAX

24-Hour AutoFax 916.753.0618. Call from your FAX. Request catalog #18.

#107

FEAT'URES

14 Speed up Your M68040 with an Interleaved Bursting EPROM Interface

26 Designing with FPGAs

36 Loader 31: A Pseudo EPROM Emulator

FEATURE ARTICLE

Ron **Stence**

Speed upYour M68040 with an Interleaved Bursting EPROM Interface

Looking to improve the performance of your latest MC68040 or MC68060 design without locking yourself into a sole source of specialized memory devices? Explore one alternative that uses standard EPROMs.

An interleaved EPROM memory interface provides an impressive price/ performance ratio for computer systems that require high-performance, EPROM-based program storage. During this article, I will explore the details of implementing an interleaved EPROM memory for systems based on the M68040 family of devices and the MC68060 processor. Since the MC68040, MC68LC040, and MC68EC040 are each pin, bus, and integer unit compatible, my discussion will apply to each of these devices. These processors will be referred to as an M68040 integer unit or as an M68040 processor.

In a design based on a 25MHz M68040 integer unit, utilization of the bursting capabilities causes a significant improvement in performance without more than an incremental increase in the overall system cost. Recently, I completed an evaluation of various memory systems capable of supporting the burst mode of the M68040 processors. Currently, some ROM manufacturers are focusing development efforts toward creating and marketing bursting EPROM technology. The cost for these devices is often prohibitive, while the memory

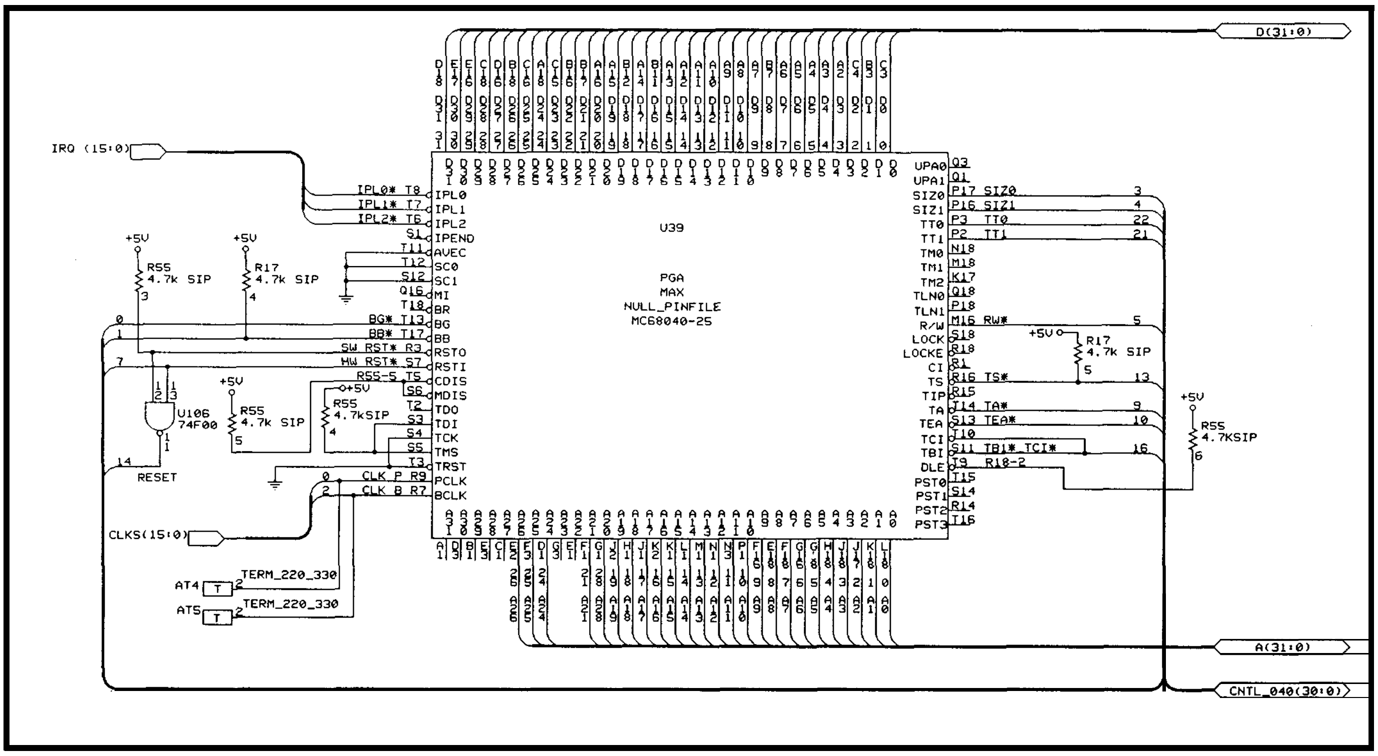


Figure 1a—At the core of the *sample design* is a Motorola *MC68040* processor. Performance can be improved by using an interleaved bursting EPROM interface.

size is limited when compared with other EPROM devices. Costs are also high because second sources for bursting EPROMs are not typically available. In this article, I will address other methods that can be used to create burst-enabled memory and provide data to support the performance improvements you may expect from an interleaved-based design.

CONSIDERING THE ALTERNATIVES

Several types of memory interfaces can be used to interface processors with today's memory devices. Most of these techniques are well known since EPROM technology has not changed much in the past few years. Notable changes in this arena have primarily focused on increasing memory size and decreasing the access time. Recently, several EPROM manufacturers have begun releasing bursting EPROMs. Designers should be careful about specifying the current offering of bursting EPROMs into their products due to the lack of second sources, limited memory size, and higher costs of these devices.

Instead of using bursting EPROMs, I have designed a two-way interleaved EPROM system which provides a burst-enabled memory system. This system can be expanded to create a four-way interleaved EPROM interface, which requires four banks of EPROMs. The more traditional linear pattern uses a single bank and does not allow for bursting. A third nonbursting alternative worthy of consideration is to use a single EPROM and an MC68 150 dynamic bus sizer to build the 32-bit word.

Bursting refers to a method used for memory accesses and also to a microprocessor's ability to make multiple accesses quickly. A burst begins with a read request to a single address that is referred to as the ***initial seed address.*** A memory system capable of bursting should provide a significant improvement in the access time of subsequent accesses after the initial seed has been accessed. Bursting may not be able to provide a significant improvement in performance if the information, data, or instructions are highly fragmented or if a zero-wait-state memory system is available. Bursting will have the greatest positive impact when all memory references are linear and then where the access penalty is large for the initial access and minimal for each subsequent burst access.

Typically, all data and instructions benefit from bursting. However, when data or instructions are highly fragmented, bursting can have a negative effect. For example, if a pointer is generated to read a single value from a memory location, then the other three accesses may never be used. In this situation, the unused cycles can stall the integer unit while the bus could have been used for other required memory accesses. When a zero-wait-state memory system is used, the M68040 integer unit will perform as well as, or better than, when the bursting function is used.

BURSTING THE M68040

The M68040 integer unit has a Harvard-style architecture that allows the six-stage pipeline and two integrated on-chip caches to operate independently. The M68040 processor has an integrated on-chip 4K-byte instruction cache and a 4K-byte data cache. Both caches are four-way-set-associative caches composed of four 32-bit words per cache line.

The relatively large instruction and data caches can each sustain approximately 90% hit rates. The hit rate varies depending on the instruction and data mixes. With a bursting

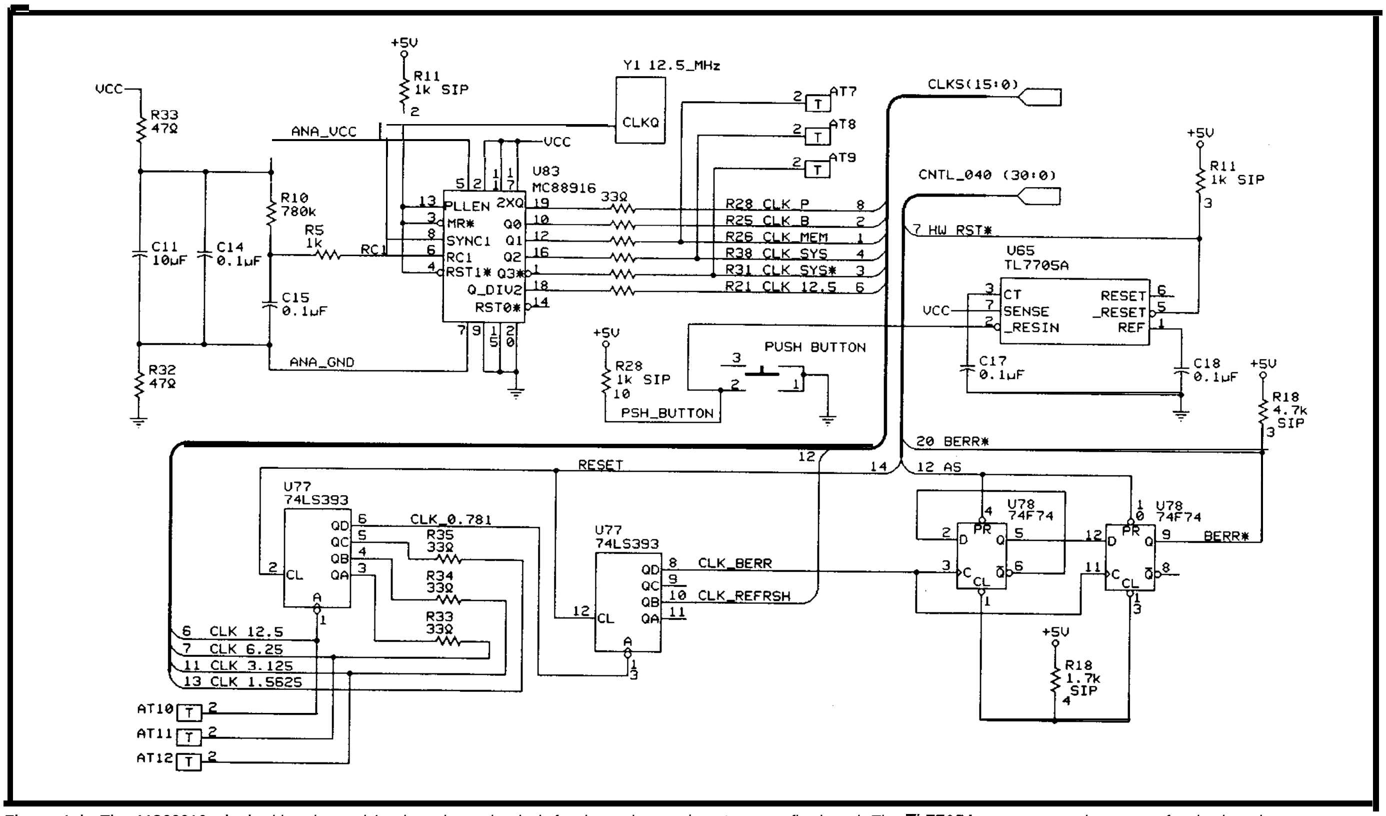


Figure 1 b--The MC88916 clock *driver is used to skew the main clock for the various subsystems on fhe board. The TL7705A generates a clean reset for the board.*

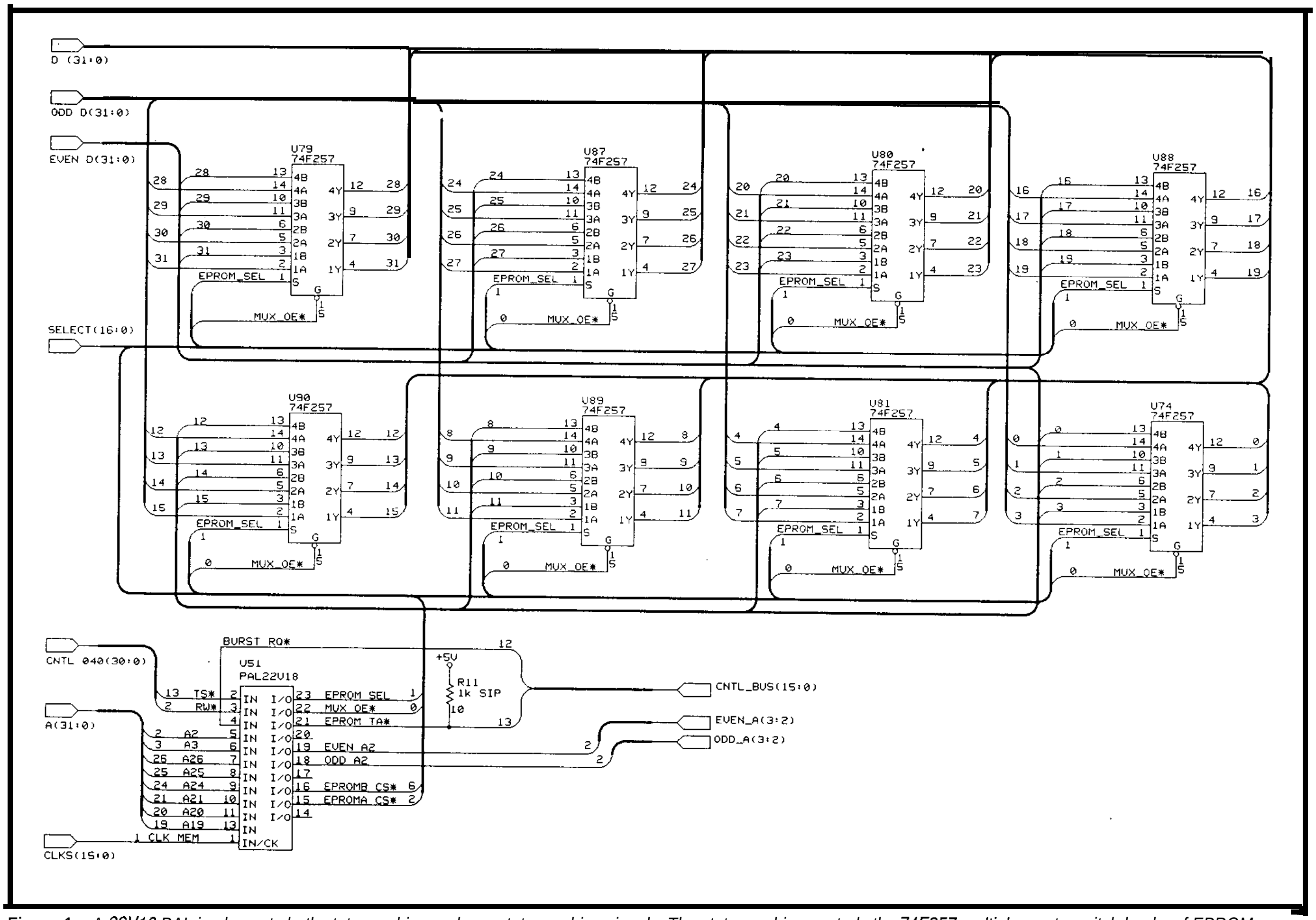


Figure 1 *c-A 22V10 PAL implements both state machine and non-state machine signals. The state machine controls the 74F257 multiplexers to switch banks of EPROMs.*

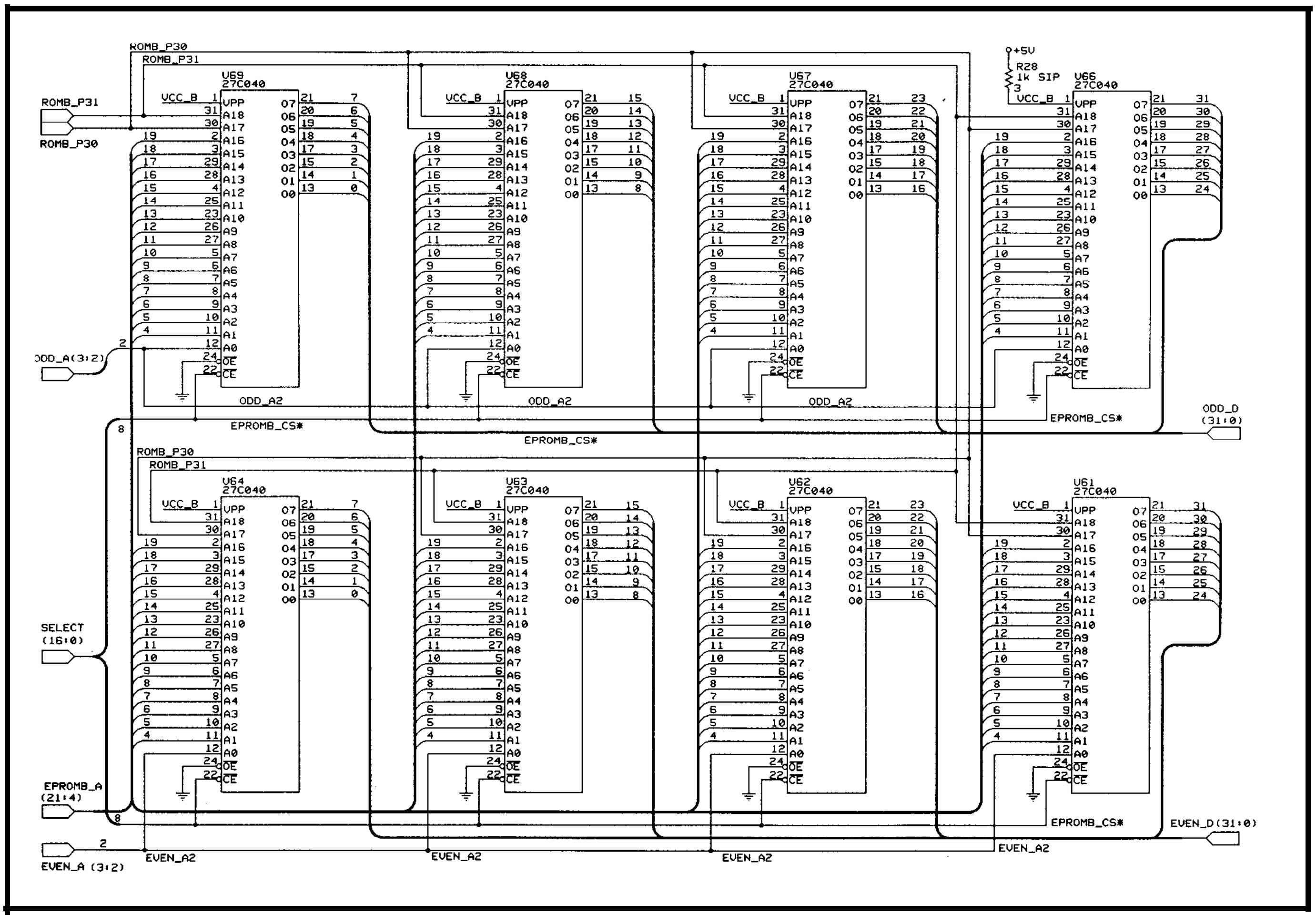


Figure 1a—Two banks of off-the-shelf 512K-byte EPROMs form the core of the memory section.

EPROM interface, the penalty for an instruction or data cache miss is minimized. This is due to the significant reduction in total bus traffic. With the reduction, the instructions or data that are required to keep the M68040 busy can be fetched from memory in a minimal amount of time.

The M68040 integer unit will access four 32-bit words for each single burst sequence. The internal caches are organized to store four 32-bit words on a single cache line. The M68040 integer unit will always begin with the access that will contain the data or instruction for which the integer unit is waiting. A bursting sequence will always begin on a hex address ending with any of the following values: $0, $4, $8, or $C. Hardware is required to increment address bits A2 and A3 to obtain the additional three addresses. For example, if hex address $0000 1008 is the initial seed address, the processor will supply the seed address $0000 1008. External hardware will be required to generate and supply the “artificial addresses” $lOOC, $1000, and $1004.

The M68040 has a nonmultiplexed address and data bus, reducing the difficulty and improving the efficiency of the interleaved design. For example, a 25MHz bursting interface with a 5: 1:3: 1 (five-clock initial, one-clock second access, three-clock third access, and one-clock final access) profile can be done by using eight 120-ns EPROMs and eight multiplexers. The schematic in Figure 1 demonstrates how to implement this kind of memory system. No additional address latches are required.

Due to the relatively high-speed bus on the M68040 integer unit and the slow turn-off time of EPROMs, bus contention is highly possible. As a reminder, recall that bus contention occurs when two devices are attempting to drive the bus at the same time. Bus contention can produce three problems: increased system noise, an increase in the total system power, and system failure when the bus drivers blow out.

Memory Access	Total Number of Clocks	M68040 Performance
2:2:2:2 No Burst	8 clocks	100.0%
4:1:1:1	7 clocks	99.7%
4:1:2:1	8 clocks	97.9%
5:1:1:1	8 clocks	96.7%
4:2:2:2	10 clocks	93.4%
5:1:3:1	9 clocks	92.9%
5:2:2:2	11 clocks	90.1%
3:3:3:3 No Burst	12 clocks	89.8 %
5:3:3:3	14 clocks	83.4%
4:4:4:4 No Burst	16 clocks	80.2%

Table 1—*A whole range of bursting access* arrangements is *possible.* Which you *choose* depends *on your* design.

PERFORMANCE CONSIDERATIONS

The 5:1:3:1 bursting access will yield about 93% of the maximum sustained performance available from the

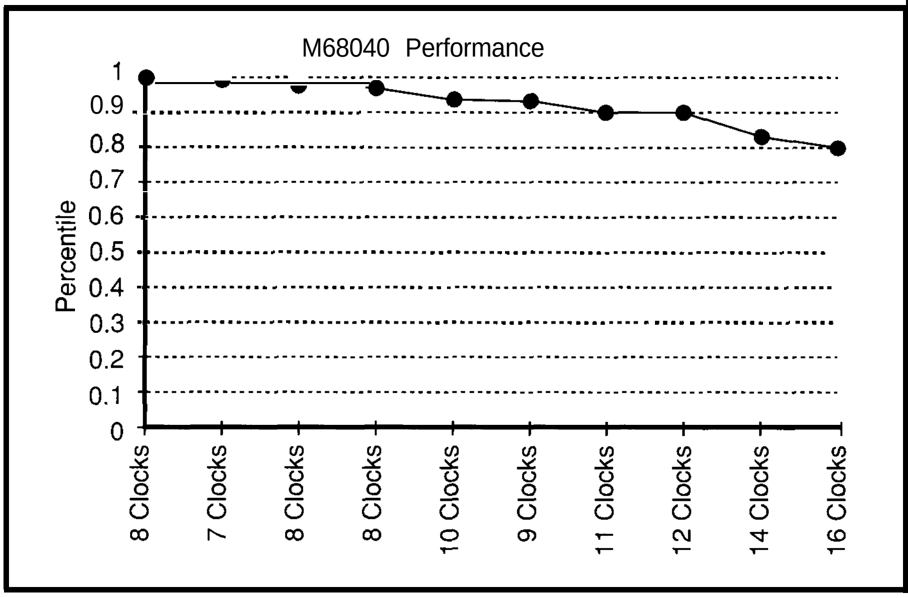


Figure 2-A 2222 *with no burst yields the* **best** performance *while the performances of other ratios and numbers of clocks fall off slightly.*

M68040 integer unit. See Table 1 for the memory interface derating chart. A graphic representation of the information contained in columns two and three of the table is shown in Figure 2.

Three points should be considered in making the entire burst access as efficient as possible. The most important part of the bursting sequence is the first access. Insertion of an additional clock cycle will typically incur a 3% degradation in performance.

The next most critical point is the second access. Often the M68040 integer unit will be starving for the first access. Once that has been accomplished, the M68040 integer unit will typically accept the second access into the first stages of the instruction pipe. When the second access is delayed by too many clocks, a *bubble* can occur in the instruction pipe. This will degrade the possible performance of the machine.

The third point to be considered is the total number of clocks for the complete burst transfer. The third and fourth transfers can degrade the possible performance by one or two percent for every additional clock cycle that is added. Typically, this is caused by stalls in data or nonsequential instruction fetches or data writes that have filled the write-back buffer.

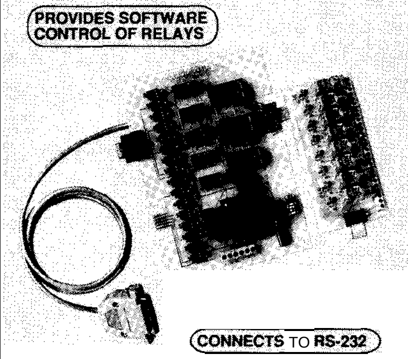
The M68040 integer unit is less susceptible to memory degradation than many other processors available today. This is due to three factors: the M68000 processors have a relatively high code density; M68000 code is typically half the size of a RISC processor; and a RISC processor is a load-

M86040 A3	Address A2	Even Bank A2	Odd Bank A2	Multiplexer Select
0	0	0	0	Even Bank
		1	1	Odd Bank
		0	0	Even Bank
		1	1	Odd Bank
0	1	1	0	Odd Bank
		0	1	Even Bank
		1	0	Odd Bank
		0	1	Even Bank
1	0	1	1	Odd Bank
		0	0	Even Bank
		1	1	Odd Bank
		0	0	Even Bank
1	1	0	1	Even Bank
		1	0	Odd Bank
		0	1	Even Bank
		1	0	Odd Bank

Table **2**—*The interleaved EPROM interface accesses two banks of memory: an even bank and an odd bank. Depending on address bit A3 from the processor, the interface will access the even or odd bank first, followed by the* opposite *bank.*

RELAY INTERFACE

PROVIDES SOFTWARE CONTROL OF RELAYS

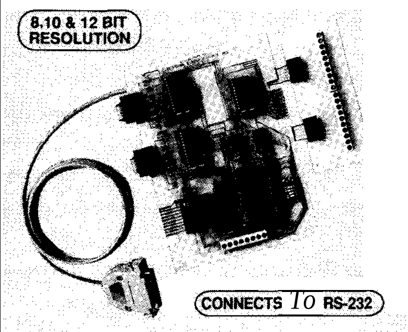


CONNECTS TO RS-232

AR-16 RELAY INTERFACE .... $ 99.95
Two 8 channel relay output ports are provided for control of up to 16 relays (expandable to 128 relays using EX-16 expansion cards). Each relay output port connects to a relay card or terminal block. A variety of relay cards and relays are stocked. Call for more info. RS-422 available (distances to 4,000 feet). PS-4 port selector may be used to control satellite AR-1 6 interfaces. (up lo 15.384 relays)
RD-8 REED **RELAY CARD (8 relays, 10 VA)..........$ 49.95**
RH-8 RELAY CARD (10 amp SPDT 277 VAC).......$ 69.95
EX-16 RELAY **EXPANSION CARD (16 channel)...$ 59.95**

ANALOG TO
DIGITAL

8,10 & 12 BIT RESOLUTION



CONNECTS To RS-232

ADC-16 A/D CONVERTER (16 channel, 8 bit)..$ 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 32 analog or 128 status inputs using the AD-16 or ST-32 expansion cards. 112 relays may be controlled using EX-16 expansion cards. Analog inputs may be configured for temperature input using the TE-8 temperature input conversion. RS-422 available. PS-4 port selector may be used to connect satellite ADC-16 interfaces (up to 4,096 analog inputs/16,384 status inputs and 14,336 relays). Call for info on 10 & 12 bit converters.
(terminal block and cable sold separately)
ST-32 STATUS EXPANSION **CARD.......................$** 79.96
Input on/off status of relays. switches, HVAC equipment, thermostats, security devices. smoke detectors and other devices including keypads and binary coded outputs. Provides 32 statue inputs (opto isolators sold separately).
TE% TEMPERATURE INPUT CONVERSION $49.66
Includes 8 temperature sensors 8 terminal block. Temperatun range is miminus 40 to 145 degrees F.
PS-4 PORT **SELECTOR** (4 channels RS-422) . . . **... .$ 79.95**
converts an RS-232 port into 4 selectable AS-422 ports.
TOUCH TONE **DECODER** end other serial interfacing products available. Call for free information packet.

- FULL TECHNICAL SUPPORT.. .Provided over the telephone by our staff. EACH ORDER INCLUDES A FREE DISK WITH PROGRAMMING EXAMPLES IN BASIC, C AND ASSEMBLY LANGUAGE. A detailed technical reference manual is also included.
- **HIGH RELIABILITY**...engineered for continuous 24 hour industrial applications. All ICs socketed.
- Use with IBM and compatibles, Tandy. Apple, Mac 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).

Use our 800 number to order FREE INFORMATION PACKET. Technical Information (614) 464-4470.

24 HOUR ORDER **LINE (800) 842-7714**
Visa-Mastercard-American Express-COD

International & Domestic FAX (614) 464-9656
Use for information, technical support & orders

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

#109

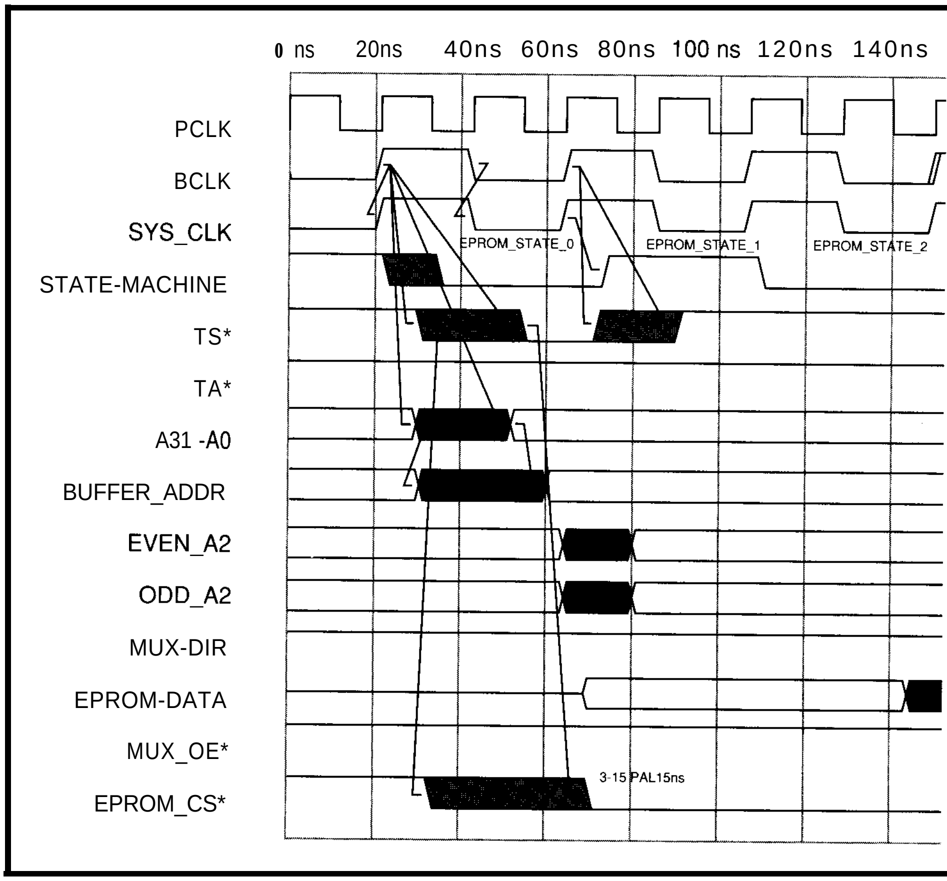


Figure 3-Precise timing of the interleaved EPROM interface is critical for successful operation. The state machine in the PAL handles all the tricky timing sequences.

store based architecture, which does not have the ability to modify memory directly.

On a RISC processor, all instructions are a fixed length, resulting in wasted bits in the instruction fields. M68000 instructions can be 16 bits or larger, depending on the addressing mode or immediate data fields that are included in the instruction. This results in very dense instruction fetches and more efficient utilization of the instruction cache. The second factor is the relatively large instruction and data caches integrated on-chip. The third factor is the bursting interface coupled to the six-stage integer instruction pipe.

When a burst is initiated, a potential data write is delayed while the instruction(s) are fetched. Because data writes can be delayed, the M68040 can fetch instructions or data that are needed to keep the instruction pipeline full. The M68040 integer unit requests the required instruction first and bursts in the remaining three fetches. Typically, the M68040 integer unit will need the next instruction within one to four clocks after fetching the first instruction.

After bursting, the M68040 integer unit has four or more clock cycles when the integer unit will not require the data bus for instruction fetches. This allows the processor to update memory or fetch new data. The M68040 integer unit will use 50–90% of the total available bus bandwidth. The lower bus utilization allows other devices such as DRAM refresh, DMA, or slow peripherals to take advantage of the available bandwidth without impacting performance.

The interleaved EPROM interface accesses two banks of memory-an even bank and an odd bank. Depending on address bit A3 from the M68040 processor, the interleaved EPROM interface will access the even or odd bank first, followed by the opposite bank. See Table 2 for the incremental burst-sequence decode. The interleave control PAL decodes the state of the M68040 integer unit's address bits A3 and A2 to decode which bank of memory (the even or odd bank] should be selected for the M68040 integer unit's data bus. During a bursting sequence, both banks of memory are each accessed twice. The four 32-bit words fill a cache line. The interleaved control PAL increments the even A2 and the odd A2 signals to provide the two accesses with different addresses for each EPROM bank. The computation of the timing required for the interleaved EPROM memory system is greatly simplified by the synchronous bus on the M68040 integer unit.

TIMING CONSIDERATIONS

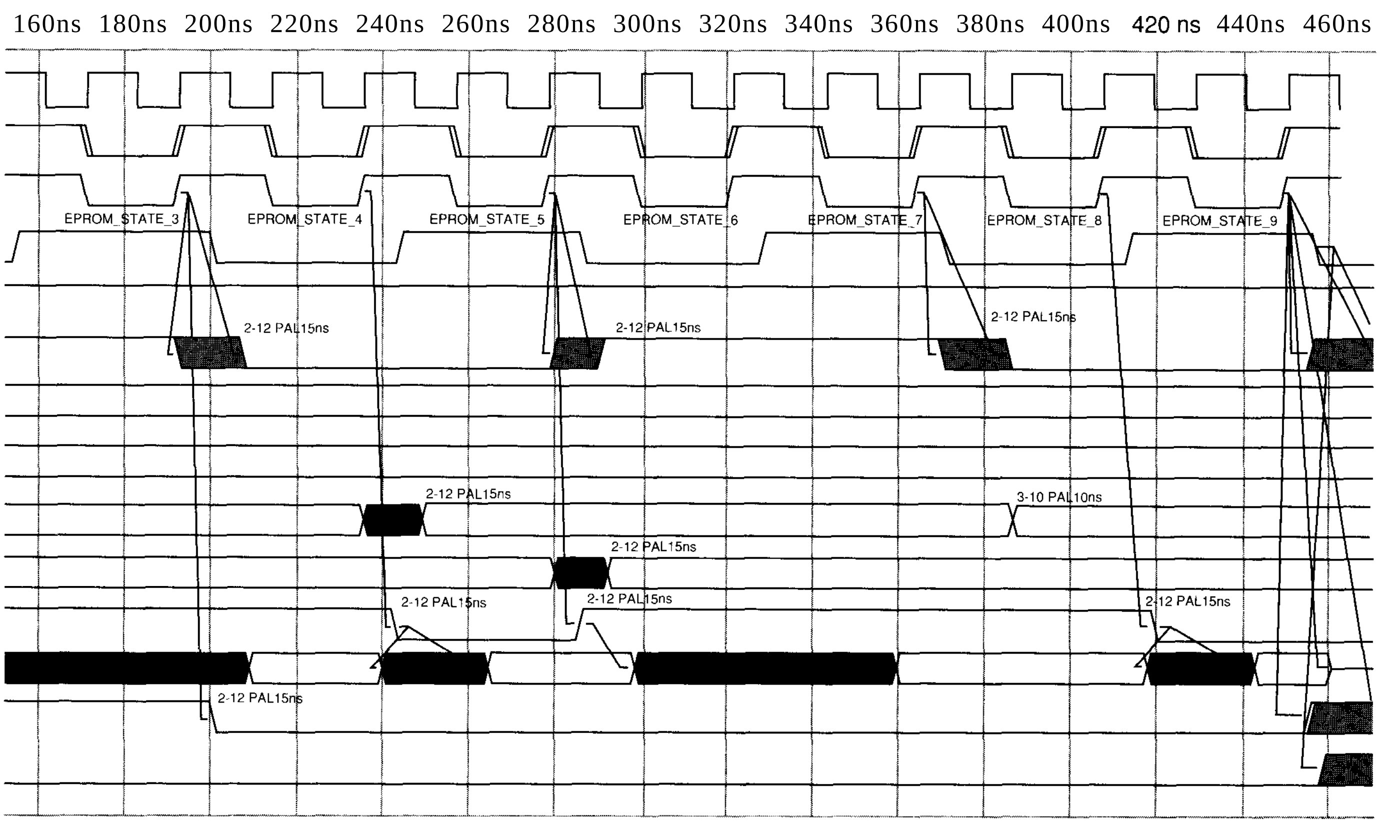
The first access begins at the rising edge of the BCLK. The signal TS* is asserted for 10 ns or more. This condition causes the state machine in the interleaved control PAL to go to state ES1. Assertion of the EVEN_A2 and ODD_A2 to the correct logic levels occurs between 2 ns and 15 ns after the rising edge of the BCLK when a 15 ns PAL is used.

$$t_{AVAIL} = (N - 1) \times t_1$$
$$t_{Rq} = t_{PALa} + t_{EPROMa} + t_{74F257P}$$
$$t_{VALID} = t_{AVAIL} - t_{Rq} - t_{SKEW} - t_{SETUP}$$

where:

t_{AVAIL} = total time available
t_{Rq} = time required by interface logic and memory
t_{VALID} = time available remaining
N = number of clock cycles
t, = time period of the bus clock
t_{PALa} = PAL logic access time to valid outputs (15 ns)
t_{EPROMa} = EPROM access time to data valid (120 ns)
$t_{74F257p}$ = maximum propagation delay for 74F257 (7 ns)
t_{SKEW} = maximum clock skew
t_{SETUP} = maximum setup time on M68040 (5 ns)

Computation of equation t_{AVAIL} is 160 ns when a 25-MHz M68040 integer unit with a five-clock initial access is used. Computation of equation t_{Rq} is 142 ns and t_{VALID} is 18 ns (specification 16 of the M68040). The hold time requires that data be held valid for a minimum of 4 ns after the rising edge of BCLK.



ECAL Universal Assembly Language Development System

- ▶ Alternative to Real-Time Emulator
- ▶ Support for 805 1.8096, and 186
- ▶ Support for 170+ additional processors
- ▶ User control of syntax and instructions
- ▶ Extremely fast assembly--? Kbytes/sec
- ▶ Integrated split-screen editor or command-line assembly supported
- ▶ Integrated linker/loader
- ▶ Instruction trace and I/O windows
- ▶ Monitor and RS-232 corn. windows
- ▶ Single micro processor versions available
- ▶ Optional EPROM emulator and programmer
- ▶ Source-level debugger

Ordering Information

- 05-0200-010 ECAL OAS
- 05-0200-020 ECAL with EPROM Emulator
- 05-0200-XXX ECAL Single Processor

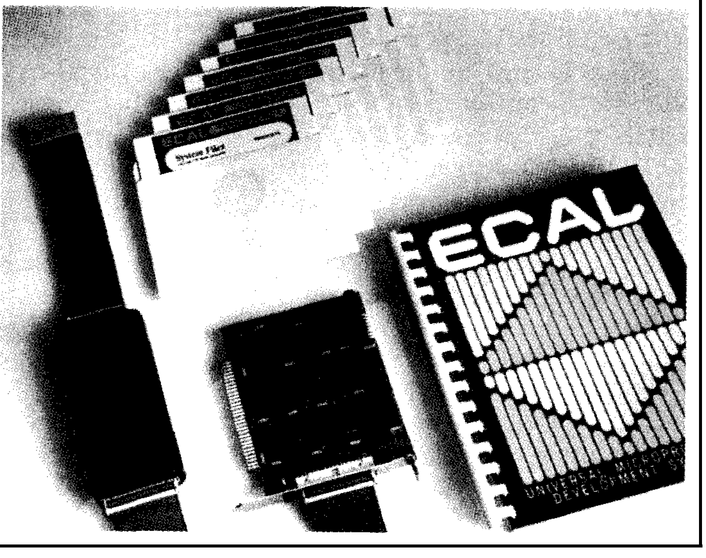
Product Information

ECAL is a complete assembly-language development system that provides all the tools needed to assemble, lmk, load. run, and debug your project for over 170 processors. By using user-editable control files, the ECAL macroassembler in its full configuration can handle 4-,8-,16-, or 32-bit microprocessors with unsurpassed speed and consistency.

Using the familiar DOS-bated text windows. you can edit, assemble. set breakpoint,. trace execution, watch registers and I/O, and communicate with your target's serial port in separate closable windows. If you prefer to use other tools, with a few keystrokes. ECAL will incorporate your previous work into its consistent and intuitive environment.

The free ECAL evaluation program features all of the ECAL tools for all ot the supported micros. giving you a true sampling of ECAL development cycle (source and object length limited).

VAIL Silicon Tools sells and supports ECAL and can bundle ECAL with additional hardware and software to satisfy your need for economical project development tools.



Contact

Vail Silicon Tools
692-A S. Military Trail
Deerfield Beach, FL 33442
Tel: (305) 570-5580
Fax: (305) 428-1811

Vail Silicon Tools

#110

$t_{HOLD} = t_{74F257h} + t_{EPROMh} + t_{PALh} - t_{SKEW}$

where:

t_{PALh} = PAL logic minimum propagation time (2 ns)

t_{EPROMh} = EPROM hold time (0.0 ns)

$t_{74F257h}$ = minimum propagation delay for 74F257 (2 ns)

Calculation of equation t_{HOLD} is 0.0 ns for all four accesses. Refer to the hold time timing diagram, shown in Figure 3 for additional information. The 0.0 ns of margin on the hold time does not cause a problem when the following considerations are taken into account.

First, the EPROM manufacturers specify 0.0 ns hold time from chip select negation to data invalid. This is not very realistic and some EPROM vendors are beginning to specify 4 ns (or more) hold time. Second, the clock driver should be placed near the M68040 integer unit to reduce transmission line effects and EMI noise. This results in the BCLK trace to the M68040 integer unit being significantly shorter than the trace to the interleaved control PAL which adds to the total data hold time.

The clock driver chosen can make a significant difference in the clock skew time. When using a MC889 15 or MC88916 clock driver, the output chosen can further reduce the skew between the BCLK and the clock pin of the interleaved control PAL. For example, the MC88915 clock driver's Q4 clocks 34 picoseconds or more (up to 275 ps) before QO. When the Q4 output from the MC88915 is used to drive the M68040 BCLK pin and the QO is used to drive the interleaved control PAL, the BCLK will always lead the PAL logic output. When using the MC88915 to drive the system clock, the clock skew becomes 0.034 ns positive. If the M68040 integer unit cannot be used with the MC88915 or MC88916 in this configuration, a delay can be introduced into the interleaved control PAL clock. A typical delay device might be a MC74F08, where the low-to-high specification is 3.0 ns minimum and 6.6 ns maximum. The interleaved control PAL clock should not exceed an 8.7-ns delay because the interleaved control PAL could miss the TS • signal from the M68040 integer unit.

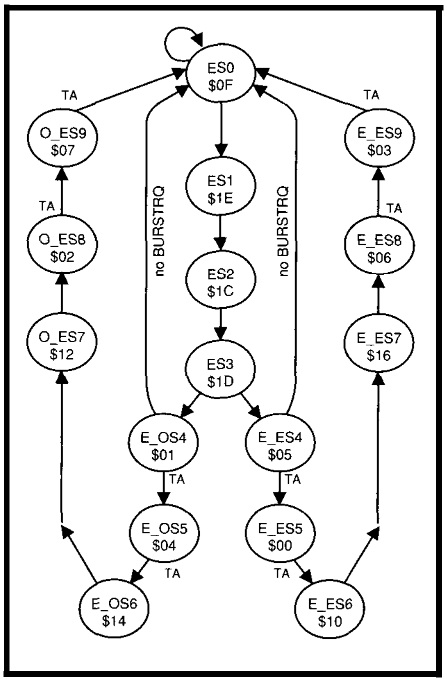


Figure 4—*Interleaved* ***EPROM*** *State* Machine ***Diagram*** *for the M68D040/LC040/EC040*

Finally, the trace length and capacitance of the control signals and data bus need to be at a minimum to require any calculations to be performed on the trace or wire delay. Furthermore, the trace and device capacitance contribute to the total hold time. The hold time of 0.0 ns should not be a problem when a high-resolution clock driver is used. The maximum clock skew should never exceed 1 ns or BCLK will lead the interleaved control PAL clock.

The second access has one additional clock period to access the EPROM. The second access setup time (t_{VAIL}) is 240 ns and the t_{VALID} is 63 ns. An additional quick check should be performed on t_{HOLD} when the multiplexer is changed from one input to the other. The minimum multiplexer select time is 3 ns. The interleaved control PAL clock-to-state change is 2 ns. When using the MC88915 or MC88915 clock driver described above, the clock skew is negligible and the t_{HOLD} time is 1 ns.

The third access has timing that is similar to the first and second accesses. The t_{Rq} for the third access is 142 ns, so t_{VALID} is 18 ns. A similar argument can be made for the fourth and final access.

THE PROCESSOR'S PAL

The PAL equations to control the interleaved EPROM interface fit into a single 22VIO device. The code used to program this device is shown in Listing 1. The inputs to the PAL are a 25-MHz system clock, TS*, R/W*, and address signals from the M68040 processor. The last input is a combination of several signals to create the BURSTRQ' signal. The BURSTRQ • signal is asserted low when the M68040 processor attempts to burst, indicated by the assertion of SIZE1 and SIZE0 signals to a high logic level. BURSTRQ* can be derived inside the 22V10 with an equation, or a 74F00 can be used to conserve input pins.

The outputs of the interleaved EPROM PAL are in two groups. The first group is the non-state machine outputs and the second group contains the state machine outputs. The signals in the non-state machine group are the EPROM chip select (EPROM_CS*) and the even and odd address bit 2 (EVEN_A2 and ODD_A2). The signal EPROM_CS * will assert when the address is valid and the TS* signal is asserted. The EPROM_CS* signal remains asserted until the state machine returns to the ES0 or the starting state. The chip select signal will assert only during a read operation. A write causes the EPROM_CS' signal to negate and the state machine to go to ESO.

The EVEN_A2 and ODD_A2 signals begin the toggling process based on the original value of the A3 and A2 address bits from the M68040 processor. For this discussion, an address will be considered "even" when the address from the M68040 processor decodes to the even bank first. An "odd" address will decode to the odd bank first. All first accesses beginning with a $0 or $C will begin with the even bank. All first accesses beginning with a $4 or $8 will begin in the odd bank of EPROM. For example, the address $0000 0004 causes the first access to occur in the odd bank. If the first access were memory location $0000 OOOC, the access would be even.

Listing I--The *22V10 PAL handles both state machine and non-state machine signals. It contains fhe main control logic for the interface.*

```
MODULE  INTERLEAVED-EPROM:

flag '-r3'

TITLE '  Interleaved EPROM PAL Eq.
         Ron Stence     Motorola Inc.'

inter device 'p22v10';

"-----------------"
"DEFINE INPUT PINS "


clk         pin 1;          "25MHz system clock
~ts         pin 2:          "68EC040 transfer start
read        pin 3;          "68EC040 read-write
~burstrq    pin 4;          "68EC040 wants to burst
a2,a3       pin 5,6;        "address bus
Address     pin 7,8,9,10,11; "address bus

"-- ---------------"
"DEFINE OUTPUT PINS "


~eprom_cs   pin 16;         "EPROM Chip Select
e_a2        pin 19;         "even EPROM A2
odd_a2      pin 18;         "odd EPROM A2
q1,q0       pin 17,20;      "state machine outputs
~eprom_ta   pin 21:         "EPROM Transfer Acknowledge
~mux_oe     pin 22;         "EPROM data MUX output enable
mux_sel     pin 23;         "mux sel =1:Even EPROM
                            "mux sel =O:Odd EPROM


"DEFINE CONSTANTS "
"-----------------"

X,C,Z,K  =.X.,.C.,.Z.,.K.:

"---------------------"
"DEFINE INTERNAL NODES  "


RESET node 25:  "A Write to EPROM will Reset this state machine!


"DEFINE VECTORS "


eprom_s = [~eprom_ta,~mux_oe,mux_sel,q1,q0];
 ES0     = [1,  1,  1,  1,  1];" ^h1F
 ES1     = [ 1,  1,  1,  1,  0];" ^h1E
 ES2     = [ 1,  1.  1,  0,  0];" ^h1C
 ES3     = [ 1,  1,  1,  0,  1];" ^h1D
 E_ES4   =    [ 0, 0, 1, 0,  1];" ^h05, assert TA to even EPROM
 E_ES5   =   [0,0,0,0,     01:" ^h00, assert TA to odd EPROM
 E_ES6   = [ 1,  0,  0,  0,  0];" ^h10
 E_ES7   = [ 1,  0,  1,  1,  01;" ^h16
 E_ES8  = [  0,  0,  1,   1,  01;" ^h06, assert TA to even EPROM
 E_ES9   =   [0,0,0,  1,   1];" ^h03, assert TA to odd EPROM

 E_OS4   =    [ 0, 0, 0, 0,  1];" ^h01, assert TA to odd EPROM
 E_OS5   = [0,0,  1,  0,  01;" ^h04, assert TA to even EPROM
 E_OS6   = [ 1,  0,  1,  0,  01;" ^h14
 E_OS7   = [ 1,  0,  0,  1,  01;" ^h12
 E_OS8  = [  0,  0,  0,   1,  01;" ^h02, assert TA to odd EPROM
```

(continued)

The PAL equation for EVEN_A2 begins with the signal asserted to a low logic level when the address is "even." The basis of the PAL equation for the signal EVEN_A2 is an exclusive-OR. The signal EVEN_A2 is used for the first and third accesses when the address is "even." Therefore, EVEN_A2 will be toggled immediately following the first access to allow the even EPROM bank to begin the next access. Once the third access has been terminated, the signal EVEN-A2 becomes a "don't care." When an "odd" access is initiated, the signal EVEN_A2 will begin with a high logic level and the toggling will begin at the end of the second access.

The signal ODD_A2 follows the initial value of the signal A3 from the M68040 processor for both "even" and "odd" addresses. When the address is "odd," the first access will be to the odd bank. At the end of the first and third accesses, ODD_A2 will toggle. When the address is "even," the first access will be to the even bank and the signal ODD_A2 will toggle at the end of the second access.

The EPROM state machine is composed of five bits. Two bits were used to create unique or distinctive state machine outputs (Q1 and QO) so a Grey code type of count could be used. However, not all state changes are true Grey code. This does not cause a problem since all signals using the state machine outputs are registered. The next bit is the multiplexer select bit (MUX_SEL). When this bit is a logic high, the multiplexer selects the even bank. When the MUX_SEL bit is a logic low, the multiplexer selects the odd bank. The fourth bit is the multiplexer output enable bit (MUX_OE*). When this bit is asserted to a logic low, the multiplexer drives the data bus of the M68040 processor. When the MUX_OE* bit is negated to a logic high, the multiplexer becomes a high-impedance output. The final output of the EPROM state machine is the acknowledge signal to the M68040 processor (EPROM_TA*).

The output pin EPROM_TA' uses an open-collector-type output. This will reduce the number of inputs to the acknowledge PAL equation and allows other devices to be connected together to drive EPROM_TA* to a low logic level. Other signals that can be connected to the EPROM_TA* signal net are the acknowledge from the interrupt PAL or from other peripherals PALs. A pull-up resistor is required when this configuration is used.

STATE MACHINE CAVEATS

The state machine (refer to Figure 4) can be modified easily to increase or decrease the number of wait states. To increase the number of wait states, an additional state should be added before states ES3, E_ES7, and E_OS7. To decrease the number of wait states, remove states ES3, E_ES7, and E_OS7.

When changing the state machine, two points must be considered. The Grey code may need modification to minimize state changes. The Q1 and QO outputs can be modified to minimize the number of state changes without impacting the operation of the PAL equation. Second, the state machine is running ahead of the M68040 processor. For example, the EPROM_TA' is asserted during state E_ES4. However, the M68040 processor will actually accept the data and TA* signal on the rising edge of the BCLK. The state machine will exit the E-ES4 state at the same time. Therefore, the modification to the state machine should take place when the state machine is going through wait states.

USING THIS NEW TOOL

Several types of applications are well suited to an interleaved-EPROM-based design, such as laptop computers, low-power computers, embedded controllers, and laser printers. A laser-printer-based application can benefit greatly from an interleaved EPROM interface due to the cost sensitivity of the laser printer market while being driven to provide higher performance for the same system cost. Laser printers typically will not copy instructions and fonts into DRAM. The interleaved EPROM interface will provide DRAM equivalent performance without a significant increase in the cost of the system.

Listing 1-confinued

```
 E_OS9  = [ 0,  0,  1,  1,  11;" ^h07, assert TA to even EPROM

addr = [Address];

state-diagram eprom_s

state  ESO:
          if (!~ts & (addr == Address)) then ES1;
          else ESO:
state ES1:
          goto ES2;
state ES2:
          goto E3;
state  ES3:
          if (!a2) then E_ES4;
          else E_OS4;
state E_ES4:                    "Assert TA, MUX sel = 1
          if (!~burstrq)  then E_ES5
          else ESO;             "no burst
state E_ES5:                    "Assert TA, MUX sel = 0
          goto E_ES6;
state E_ES6:
          goto E_ES7;
state E_ES7:                     "Assert TA, MUX sel = 1
          goto E_ES8;
state E_ES8:                     "Assert TA, MUX sel = 0
          goto E_ES9;
state E_ES9:
          goto ESO;

state E_OS4:                     "Assert TA, MUX sel = 0
          if (!~burstrq)  then E_OS5
          else ESO:             "no burst
state E_OS5:                    "Assert TA, MUX sel = 1
          got0 E_OS6;
state E_OS6:
          goto E_OS7
state E_OS7:                    "Assert TA, MUX sel = 0
          goto E_OS8;
state E_OS8:                    "Assert TA, MUX sel = 1
          goto E_OS9;
state E_OS9:
          goto ESO;


"PAL EQUATIONS"
"-------------"

equations

RESET = !read;

e_a2 := ( ((a3 $ a2)
  & (!~ts # (eprom_s == ESO) # (eprom_s == ES1)
         # (eprom_s == ES2) # (eprom_s == ES3))
         # (e_a2 & ((eprom_s == E_ES6) # (eprom_s == E_ES7)
         # (eprom_s == E_ES8) # (eprom_s == E_OS6)
         # (eprom_s == E_OS7) # (eprom_s == E_OS8)
         # (eprom_s == E_OS9) )) # (!a3 & ((eprom_s == E_ES4)
         # (eprom_s == E_OS4))) # ((a3 !$ a2) & ((eprom_s == E_ES5)
         # (eprom_s == E_OS5))) );

odd_a2 := ( ((a3 $ a2) & ((eprom_s == E_ES4)
           # (eprom_s == E_OS4)))
           # (a3 & (!~ts # (eprom_s == ESO) # (eprom_s == ES1)
           # (eprom_s == ES2) # (eprom_s == ES3)))
           # (!a3 & ((eprom_s == E_ES5) # (eprom_s == E_ES6)
```

(continued)

Listing 1-continued

```
            # (eprom_s == E_ES7) # (eprom_s == E_ES8)
            # (eprom_s == E_ES9) # (eprom_s == E_OS5)
            # (eprom_s == E_OS6) # (eprom_s == E_OS7)
            # (eprom_s == E_OS8) # (eprom_s == E_OS9))))

!~eprom_cs = read & ((!~ts & (addr == Address) )
            # ((eprom_s != ES0) & (addr == Address)));
enable ~eprom_ta = (!~eprom_ta);

end INTERLEAVED-EPROM;
```

The interleaved EPROM interface has several positive points to be considered. The most significant is a dramatic increase in system performance while reducing bus utilization and memory latency on the M68040 integer unit. Second, the total system power can be reduced when the EPROMs do not have to be copied to a higher-speed memory system. When the EPROMs are not copied, power and money can be saved while not supporting a second memory system. Third, there will be a reduction of bus contention between the EPROMs and the M68040 integer unit. Finally, when data bus buffers are required, the change to multiplexers will have little or no effect to the cost of the system.

An interleaved EPROM interface does have several negative attributes which should be considered before using it in a design. First, there is an increased number of EPROMs included in the design. This technique should not be considered when the design will be using eight or more EPROMs. Second, there is an increased number of support devices such as multiplexers. Third, the increased complexity in the control logic could result in an increase in the board size. Finally, the potential increase in total system cost should be considered.

The change to an interleaved EPROM interface can be done with a small increase in complexity and system logic. The positive and negative attributes, such as total system size, cost, and performance levels, should be considered before changing to an interleaved interface. A requirement to make a dramatic improvement in the system performance could make an interleaved-EPROM-based design a wise solution.

Ron Stence holds a B.S. in Computer Science from the Department of Engineering at Texas A&M University. He is currently a senior systems applications designer in Motorola's 68000 marketing applications group.

I R S

401 Very Useful
402 Moderately Useful
403 Not Useful

PRO = Positively Rampant Optimization

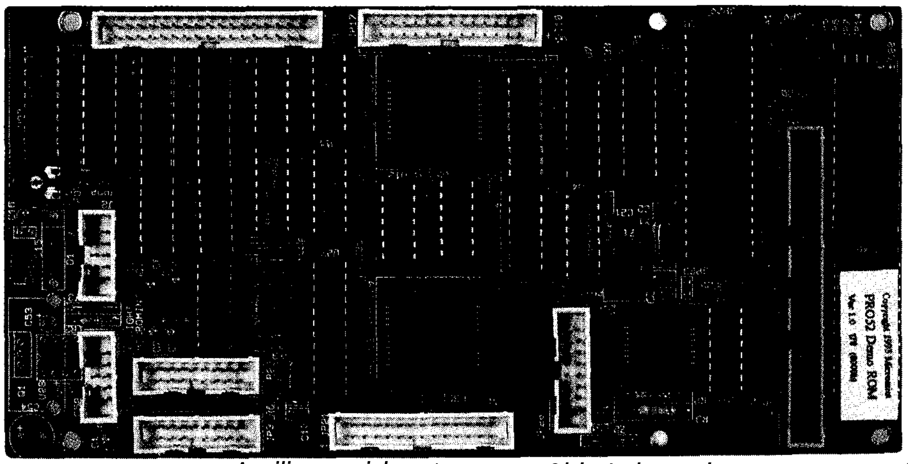
4 x 5 ***keypad input***

4 × 20 or 8 × 40
LCD display output

Hardware real-time clock
24-bit *parallel* ***TTL I/O***

CMOS *PROCESSORS*
• ***80C31/Processor*** or
. ***80C52*** with BASIC52

8/32K *RAM*



PRO31 prices start a5 low as $289 ($219/100), includes SK RAM, watchdog, 56 bits I/O, S-bit ADC, and LCD interface. ***And it only costs $10 more for PRO52!***

8/32K *ROM*

Power inputs
- ***Single*** supply *or* ***multisupply*** option
Console serial ***port***

Auxiliary serial port
24-bit *parallel TTL* ***I/O***
Hardware watchdog

&bit, I-channel DAC
8- *or* ***10-bit, 8-channel*** *A DC*

lEXPANSION HEADER
• Port 1 processor *lines:* ***8-bit 77L***
• ***Six decoded*** *I/O address* strobes
• ***Eight address lines***

MICRO MINT **PRO31/52 MICROMINT, INC.**

4 Park Street • Vernon, CT 06066 • (203) 871-6170 • Fax (203) 872-2204
in Europe: (44) 0285-658122 . in Canada: (514) 336-9426 . in Australia: (02) 888-6401 • Distributor Inquiries Welcome!

#119

Designing with FPGAs

Part 1: An Overview

FEATURE ARTICLE

Del Hatch

It started with the transistor, then led to SSI, MSI, LSI, VLSI, PALs, and now Field-programmable Gate Arrays. There is a whole range of FPGAs available on the market. How do you pick which one to use?

There is no doubt about it-today's electronic designs are more complex and contain more logic than ever before. We all like to see lots of bells and whistles in electronic products (even those we design) and this requires more and more logic to do the job.

However, this complexity causes a variety of problems. Having lots of chips in a design is expensive, and not simply the cost of the parts. They require a lot of board space and a lot of traces on a printed circuit board to interconnect them all [or sore wrists and fingers if you wire-wrap your designs). A board with a lot of logic chips is almost always current-hungry as well.

One solution to these problems is to put more and more logic onto each chip. Where only a little bit of logic is required, ***programmable array logic*** (PAL) chips can be used. The amount of standard parts that can be replaced by a typical PAL varies by size of the PAL and the type of circuitry you're trying to replace, but for most cases you can count on replacing around ten TTL chips with one medium-sized PAL. These programmable chips have proven very useful because they are inexpensive and can be programmed in the lab or at home.

MY LOGIC GROWS AND GROWS

But what do you do if you have more logic than can fit into a single PAL? Of course! You use more PALs, right? Not if you know how to use the latest in digital integrated circuit technology. The proper solution for many designs is applying a field ***programmable gate array*** (FPGA) instead of a handful of PALs. "Field programmable" means they can be configured in a lab or home workshop to perform custom functions.

This relatively new family of chips is produced by a few different companies, and can put the equivalent of hundreds of gates into a single, inexpensive chip. They are great for prototyping designs that will eventually be made in large quantities, as well as a "production run" of a single breadboard.

Many engineers, as well as people who like doing custom designs at home, may feel these chips are too exotic, or too hard to program, or too expensive to use in designs, but I have found the opposite to be true. When a moderate-to-large amount of logic needs to come together, FPGAs are a quick, easy, almost fun solution. The fun comes from seeing a design that incorporates a lot of logic run in a single chip on your board.

With the eventual goal of showing you how to make something useful with these chips, it is best to look at them in more depth. To do this, I will start by looking at the principles behind programmable logic devices, and then how some manufacturers use these concepts in FPGAs. Then I will look at other styles of FPGAs that broke from the PAL mold and use small "chunks" of logic that are connected together to form a design.

In the article that will appear in next month's issue, I will show you how to take a design from concept to finished product, with a unique desk clock as an example. But, this month I'll concentrate on the facts of the matter.

PROGRAMMABILITY

Each of these families of products falls under the more general classification of a ***Programmable Logic Device,*** or PLD for short. PLDs contain logic gates just like any other digital chip, but this logic is arranged differently. Instead of prewired, fixed paths for interconnecting the logic, PLDs use programmable links to connect gates together. In most PLDs, it is not the logic inside the devices that is changed

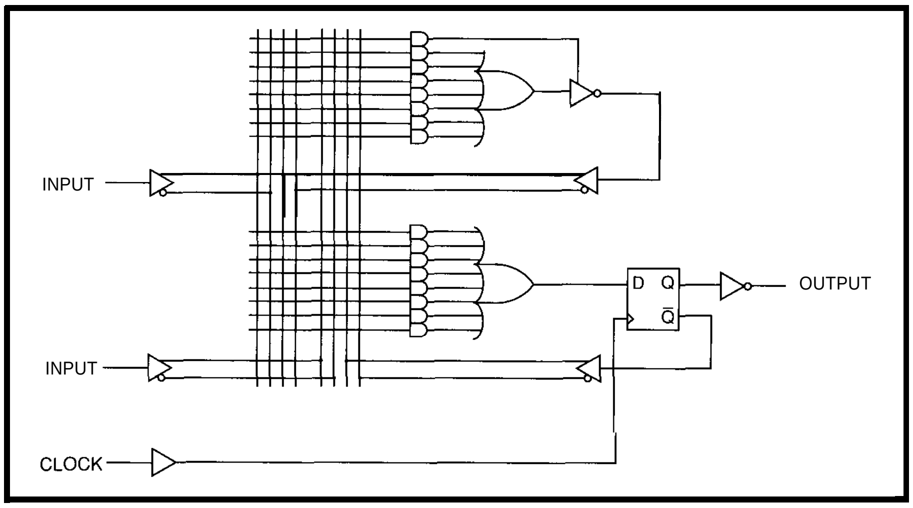


Figure 1—*This portion of a typical PAL shows the programmable connection matrix and the AND/OR logic structure.*

during programming-just the interconnections. We will later see that some FPGAs broke from this mold by storing more than just interconnection information.

The earliest and still very popular method of programming an interconnection was to blow a fuse-like connection open, so that unwanted connections are not made. When the PLD is manufactured, all interconnections on the chip are already made, and the user opens up those connections that are not wanted.

There is another method of determining how connections are made. Some PLDs use an *antifuse,* which is the opposite of a fuse, as the name implies. A connection location is manufactured in a high-resistance state (as opposed to the low resistance which an unblown fuse provides) and the user can then program it to become a low-resistance connection. In other words, no connections are premade during manufacturing. Instead, the user creates "shorts" that make the desired connections to implement the proper logic functions.

Another kind of PLD technology is similar to that used in EPROMs. These PLDs are based on EPROM technology so the connection information can be erased with ultraviolet light. This allows the device to be reprogrammed should the design change. With fuse- or antifuse-based PLDs, a mistake causes the entire PLD to become scrap. EPROM-based PLDs can be used over and over.

Another connection technology uses CMOS transistors in a pass-gate configuration. The signals pass through the connection depending on the logic level applied to the gates of the transistors. This logic level is not stored permanently in the FPGA, but instead read into the device every time power is first applied. This also means the FPGA itself is not programmed and thus another external storage device is needed.

PAL BASICS

The basic operation of a typical PAL is easy to understand. Referring to Figure 1, input signals enter the diagram from the left and are sent out on the vertical rails in their true and inverted form. Each of the AND gates (which are drawn just to the right of the wiring array) is shown with only one input, but this is only to simplify the diagram. In reality, each AND gate can operate on any of the signals that run down the vertical tracks.

The OR gates that follow the AND gates do just what they look like-they OR together each of the seven signals coming from the AND gates. In this way, logic equations can be implemented directly from the sum-of-products form.

For example, the XOR function of two inputs A and B can be written two ways:

$$Y = A \oplus B$$

or as,

$$Y = AB' + A'B$$

The second form shows how the output term is generated in a PAL-like device. One AND gate will have two inputs: the A signal and the inverted B signal. These signals are available on the vertical wiring paths as described before. The second AND gate similarly connects the inverted A input and the noninverted B input. The OR gate that follows the AND gates then takes those two terms and generates the final output Y.

The PAL quickly became the standard for programmable logic due to the simplicity and parts count reduc-

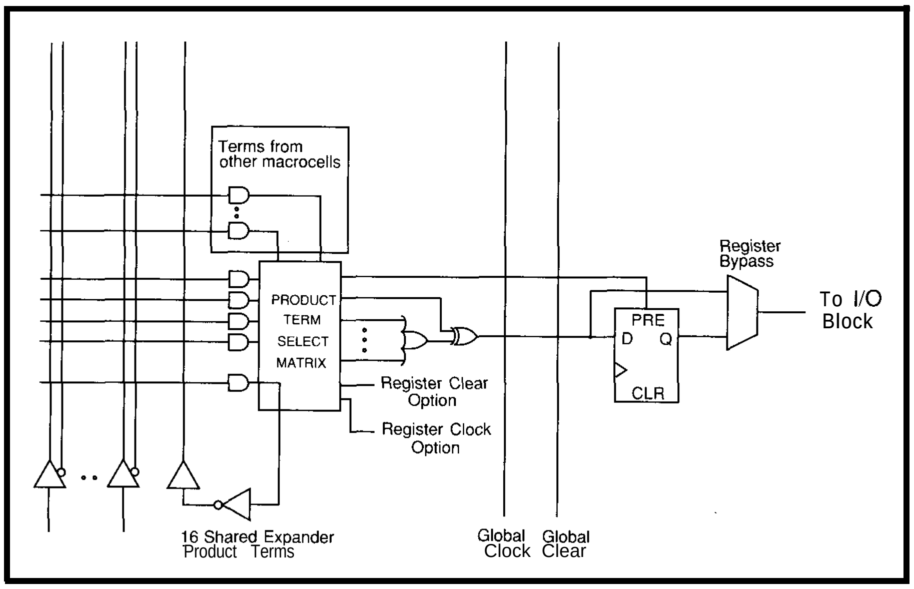


Figure 2—*The Altera macrocell has a PAL-like structure of AND/OR gates but is more flexible. Note that the flip-flop be cleared and clocked from global or locally generated signals.*

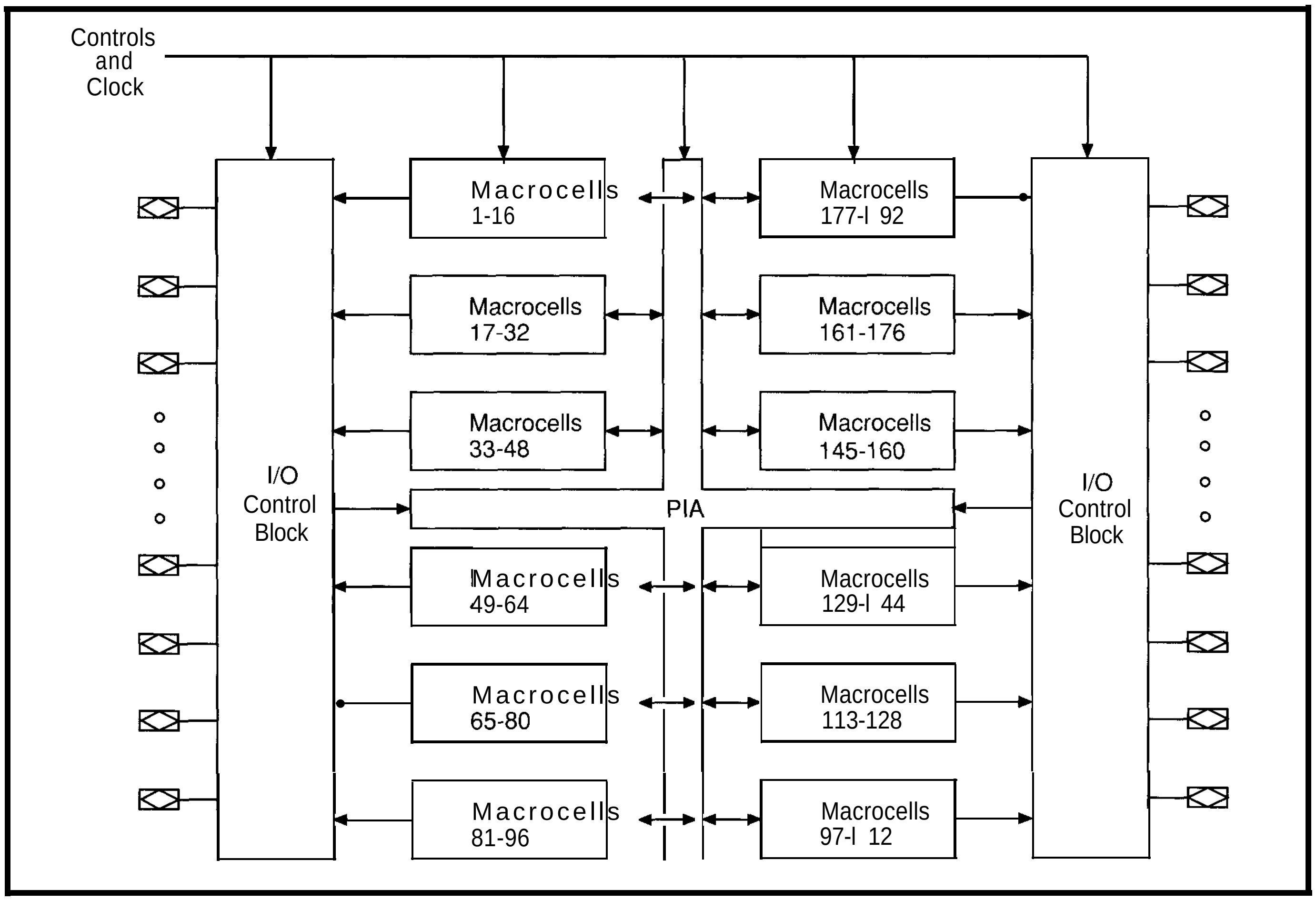


Figure 3—*The Altera* FPGAs *contain groups of macrocells and I/O blocks which communicate through the* P/A wiring resources.

tions that were possible. Another necessary part of their success was due to the software tools. It would be tedious for a human to figure out all the fuses that need to be blown, and so PAL manufacturers developed programs that would automatically create the necessary files. Without automated software tools, all programmable devices are virtually useless. A user relies so heavily on software that the tools available can often determine which device to use.

For a long time, a PAL user usually had very few options when it came to entering the design into these software tools. It usually required the logic equations to be entered. The program would then generate a fuse map that contained the information on which fuses to blow, and then a special programmer would modify (program) the chip with the proper connections.

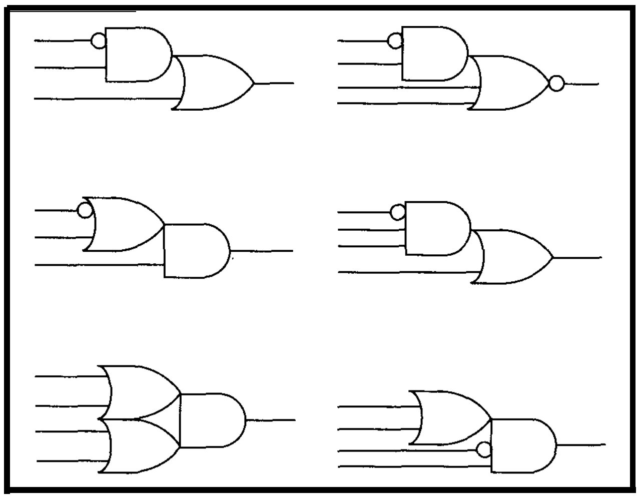


Figure 4-These *are a few typical multiple-gate functions that can be implemented in a single* ACT *1 logic module.*

ENTER THE FPGA

A major revolution began when field programmable gate arrays (FPGAs) were introduced. FPGAs allow for a lot more logic to be incorporated into one chip, thus requiring less space on a circuit board. For most people, one of the most painful things about prototyping is wiring together all the ICs. With an FPGA, the logic can often be reduced to a small number of chips.

FPGAs also allow for rapid changes in a design. Instead of rewiring a lot of logic, changes are simply made by programming another FPGA (or in the case of reprogrammable devices, simply reprogramming the same device). This reduces the stress of having to get it perfect the first time and reduces the cost when you don't. It also makes it more desirable to add functionality later because it is so painless to add logic to an already-completed design.

Like their PAL predecessors, some FPGAs use a wiring matrix that ANDs and then ORs the input signals. The most popular manufacturer of this type of FPGA is Altera. (The term "manufacturer" is not entirely accurate. Like most FPGA companies,

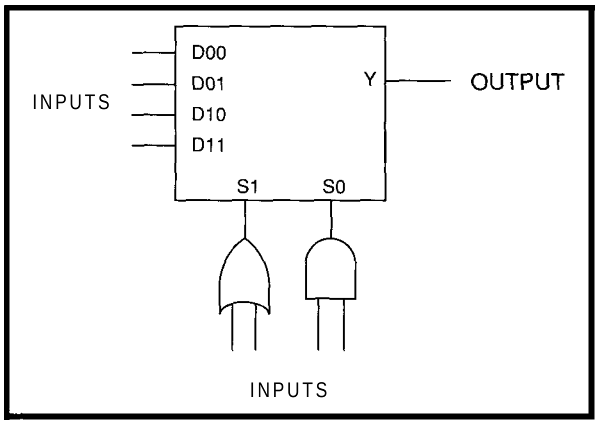


Figure 5-The *ACT 2* C-module uses a *4-input* multiplexer along with gates *on the selection pins to implement different logic functions.*

Altera designs the chips and does the layouts and another company, the "foundry," actually manufactures the ICS.)

Altera has developed a few different families of FPGAs, starting with the MAX (Multiple Array matrix) 5000 family, then continuing with the MAX 7000 family, which is currently very popular. Last year, Altera introduced the FLEX (Flexible Logic Element matrix) 8000 family, which is not structured like a PAL, but instead uses programmable blocks of logic based on lookup tables. Those readers familiar with Xilinx FPGAs will perhaps recognize the concept. I will cover this later for those who are not familiar with these chips.

I will not discuss the MAX 5000 here because it is similar to the MAX 7000 family, but not as powerful or popular for new designs. The MAX 8000 family is somewhat similar to the Xilinx family of FPGAs, which I will discuss later. For now, let's look more closely at the MAX 7000 family.

As I mentioned previously, Altera FPGAs contain groupings of logic that perform an AND-then-OR function with certain other options. Altera calls these PAL-like modules *macrocells.* Figure 2 shows what is contained in the MAX 7000 macrocell. Five product terms can be generated in the logic array, and the product-term select matrix determines if these signals go to the OR gate, XOR gate, or instead are used as register control signals.

Signal inversion can be accomplished with the XOR gate and registering of the signal is possible using the included flip-flop. The clocking for this flip-flop can be done with the global clock or from a signal generated in the logic array section of the macrocell. Similarly, the flip-flop can be cleared globally or locally.

Referring to Figure 3, 16 macrocells are combined into larger blocks called a *logic array block* (LAB). The *programmable interconnect array* (PIA) routes the necessary signals between these LABs. Because all LABs have dedicated inputs and routing resources, most signals will not travel on the PIA. Instead, the PIA is only used as necessary to share signals with the other LABs.

The entire FPGA is thus made up of interconnected LABs and one other type of block called the *I/O control block.* The I/O control blocks are used to bring in signals from the external pins of the FPGA or send signals from the logic to the pins. The block configures that pin as input, output, or bidirectional.

The structure of this chip allows for fast and predictable operation due to the uniform structure of the interconnects. You will see that with other styles of FPGAs, it is harder to know in advance what the worst-case wiring delays will be.

Altera, unlike most early PALs, chose not to rely on fuse-based technology to program the interconnections. They originally decided to use EPROM-like technologies that allow for the connections to be erased. This allows for reuse of these FPGAs. They now also offer chips that use static-RAM cell based designs, as well EEPROM technology.

DIFFERENT FPGA LOGIC STRUCTURES

Meanwhile other companies such as Actel and Xilinx (and later Crosspoint, QuickLogic, and others) took a different approach to program-

Real-Time Multitasking with DOS

for Microsoft C, Borland C, Borland/Turbo Pascal

Develop Real-Time Multitasking Applications under MS-DOS with RTKernel!

RTKernel is a professional, high-performance real-time multitasking kernel. It runs under MS-DOS or in ROM and supports Microsoft C, Borland C++, Borland/Turbo Pascal, and Stony Brook Pascal'. RTKernel is a library you can link to your application. It lets you run several C functions or Pascal procedures as parallel tasks. RTKernel offers the following advanced features:

- pre-emptive, event/interrupt-driven scheduling
- number of tasks only limited by available RAM
- task-switch time of approx. 6 µ secs (33-MHz 466)
- performance is independent of the number of tasks
- use up to 64 priorities to control your tasks
- priorities changeable at run-time
- time-slicing can be activated
- programmable timer interrupt rate (0.1 to 55 ms)
- high-resolution timer for time measurement (1 µ sec)
- activate or suspend tasks out of interrupt handlers
- programmable interrupt priorities
- supports math coprocessor and emulator
- semaphores, mailboxes, and message-passing
- keyboard, hard disk, and floppy disk idle times usable by other tasks
- interrupt handlers for keyboard, COM ports, and network interrupts included with source code
- supports up to 36 COM ports (DigiBoard, Hostess boards)
- full support of NS16550 UART chip
- fast, inter-network communication using Novell's IPX
- runs under MS-DOS 3.0 to 6.0, DR-DOS, LANs, or without operating system
- DOS calls from several tasks without re-entrance problems
- supports resident multi-tasking applications (TSRs)
- runs Windows or DOS Extenders as a task
- supports CodeView and Turbo Debugger
- ROMable
- full source code available
- no run-time royalties
- free technical support by phone or fax

RTKernel-C (MSC 6.0/7.0/8.0, BC++ 1.0/2.0/3.x) $495 (Source Code: add $445)
RTKernel-Pascal (TP/BP 5.x/6.0/7.0, SBP 6.x) $445 (Source Code: add $375)
For international orders, add $30 for shipping and handling. Mastercard, Visa, check, bank transfer, or COD accepted.

FREE DEMO DISK
Please request Info Kit !

On Time
MARKETING
Professional Programming Tools

In North **America, please contact:**
LEL Computer Systems
20 Canterbury Court
Setauket, NY 1173.3 • USA
Phone (516) 473-8119 • Fax (516) 331-0706

Outside North America, please contact:
On Time Marketing
Karolinenstrasse 32 • 20357 Homburg • GERMANY
Phone +49 40 43 74 72 • Fax +49 40 43 51 96
CompuServe 100140,633

#111

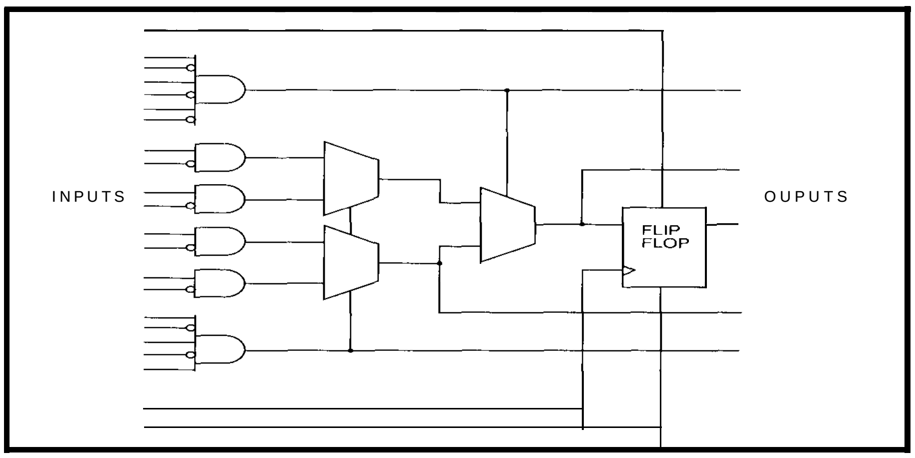


Figure 6—*This* cross-section *of a QuickLogic antifuse shows how a link is formed by burning through the silicon that separates the* two metal layers.

mable logic. They decided that the best way to implement programmable logic is to get away from a matrix of AND and OR gates and instead use smaller groups of logic that can then be wired together.

Each little logic block receives a limited number of terms and generates only one or two output terms. These blocks are then placed in a matrix pattern and programmable wiring that runs in between them connects them together to form the completed chip. Actel calls their blocks *modules,* Xilinx calls them *configurable logic blocks* (CLBs), and QuickLogic refers to them as logic cells. The main difference between them all is the complexity of the logic blocks and how many inputs can be operated on simultaneously.

ACTEL

Actel started with the ACT 1 family of FPGAs and has since come out with the ACT 2 and ACT 3 families. The biggest change occurred with the ACT 2 family, which packs more logic into each module.

The modules for the ACT 1 family are multiplexer based and can implement any AND, NAND, OR, and NOR gate possible (up to four inputs) while inverting any or all of the inputs. Each of these gates requires only one module. Certain other combinations of AND-then-OR and OR-then-AND are also possible in only one module. Some of these are shown in Figure 4. D-type flip-flops, JK-type flip-flops, and a two-input multiplexer followed by a D-type flip-flop are possible, but these require two modules each.

In the ACT 2 and ACT 3 families, the modules were modified to allow more flexibility. One of these is called *C-module* (combinatorial module) and is shown in Figure 5. Another type of module called the *S-module* (sequential module) is made of a C-module followed by a flip-flop. These two modules are supplied on the Actel FPGAs in equal numbers in an array.

A medium-sized ACT 2 part (the A1240) has 14 rows and 62 columns of these modules with up to 104 I/O lines. The I/O modules (not shown) can implement bidirectional and tristate functions as well as latching of the inputs and outputs. Registering of signals in the I/O module is only available in the ACT 3 family.

Actel chose to use antifuse technology that uses a low-resistance path where a connection is desired. In this way, the logic cells are programmed for the function they will perform and the wiring is set up to interconnect them. The drawbacks in some of the early chips concerned their reliability and speed. The connections formed during programming were not as reliable as many people wished. Also, the resistance of the connections coupled with the inherent capacitance of integrated circuit paths created a limit on the maximum speed at which the chips could operate.

As antifuse technology grew up, reliability increased and the connection resistance fell, making it possible to run the Actel chips faster and more reliably than before.

QUICKLOGIC

QuickLogic FPGAs were developed to address the need for speed. They accomplished this by using a lower-resistance antifuse than Actel. The antifuse structure is shown in Figure 6. It requires an added processing step to a standard 1 -micron CMOS gate array process involving the deposition of amorphous silicon into

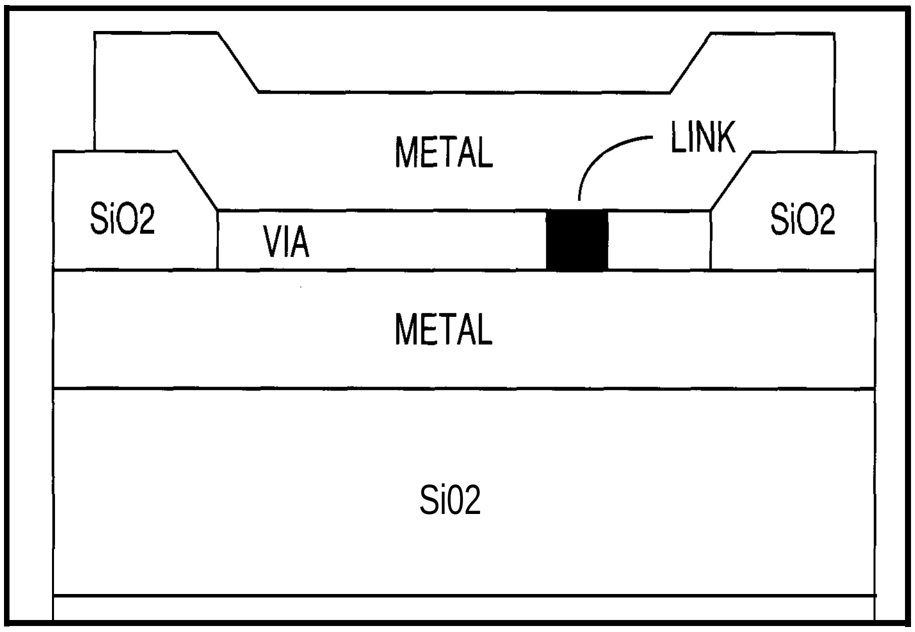


Figure 7-The *QuickLogic* logic cell uses gates and *multiplexers like Actel, but in a different structure that allows for* more inputs and outputs.

the gap between two metal layers. During programming, this layer allows for a metal-to-metal junction with a typical resistance of less than 50 ohms.

The logic module shown in Figure 7 is called a *logic cell.* This cell from the QuickLogic 1 family of FPGAs is similar in structure to the Actel S-module, but is more complex due to the addition of logic on the input to the multiplexers. It also can operate on up to 14 terms at once, although in practice this is rarely used. The size of the array of cells in a QuickLogic FPGA varies, but the FPGA with 104 I/Os has an array of 16 by 24 cells.

CROSSPOINT

Taking the concept of simple logic modules to the extreme is the transistor-pair-based module in Crosspoint's FPGAs. This module makes the structure very similar to gate arrays, where the logic is defined and implemented at the transistor level. In fact, a simple pair of p- and n-channel transistors make up a *transistor pair tile* (TPT) which is one of the two types of "tiles" that form the FPGA. A typical gate, like a 3-input NAND gate, requires four of these tiles.

The other type of tile is a *RAM-logic tile* (RLT) that is used to make memory structures, multiplexers, and exclusive OR and NOR gates.

The chip architecture is called a *channeled gate array structure.* It contains logic rows and routing channels between these logic rows where most cell-to-cell connections are made. Just like the Actel and QuickLogic FPGAs, Crosspoint uses antifuse technology for one-time programmability.

The Crosspoint FPGA structure is defined on a very fined-grained level and thus requires a lot of software intelligence to determine the routing of interconnections. At present, the logic for the designs can be entered into a PC, but the final steps require software that at present only runs on Sun or HP-Apollo workstations.

XILINX

Xilinx was one of the first FPGA companies and is still the most popular. They started with the Xilinx 2000 family and have since introduced the 3000 and 4000 families. With each generation, the logic modules (called *configurable logic blocks or* CLBs) have become more and more complex They also recently introduced a line of FPGAs which have a structure similar to the Altera FPGAs.

Figure 8 shows the configuration of a 3000 family CLB. It looks very complicated, but the basic operation is simple to understand. The combinatorial function block implements any function of the allowed variables that feed it because it is based on a lookup table. This means all combinations of input signals have been preprogrammed to give the proper output.

In many cases, the combinatorial logic block accepts five variables: A, B, C, D, and E. However, the function block may also use the output of the flip-flops (QX and QY)-but at the exclusion of other terms-for a maximum total of five terms. The combinatorial function block can also implement two different functions of four variables, as long as both functions have certain input variables in common.

An Unbiased Survey for DOS Developers.

☐ **NO!**

I don't want to find out how I can save a lot of money using ROM-DOS 5 instead of **MS-DOS®** in our 80x86 product line. I don't care if ROM-DOS 5 is compatible with MS-DOS 5 but costs much less. I like spending much more than I have to. It makes me feel like a philanthropist and besides Microsoft@ probably needs the money more than I do anyway.

☐ Yes, I want to know the facts about ROM-DOS 5. Please send me information and a free bootable demo disk to try with my software.

In the U.S.A. Call Toll Free 1-800-221-6630

or fax this coupon to (206) 435-0253.

Name ______________________

Company ______________________

Address ______________________

City ____________ State ________ Zip ________

Phone ____________ Fax ____________

307 N. OLYMPIC, SUITE 201 • ARLINGTON, WA 98223 USA • **(206) 435-8086** • FAX: **(206) 435-0253**

Datalight®

MS-DOS and Microsoft are registered trademarks of Microsoft Corporation

This can sound complex, and it is a relief to know that as with all FPGAs, a user does not have to make decisions on how the block is arranged, what variables feed it, or even what a CLB (or macrocell or module or logic cell) does! The software is designed to handle all the partitioning of the logic, the placing of the logic into different blocks, as well as doing the routing when the design is finished.

Like PALs and other FPGAs, there is a range of output options in each Xilinx CLB such as registered outputs [putting the output of the combinatorial function block through a flip-flop), an inverted clock (so the outputs are registered on the falling edge of the clock), asynchronous resets for the flip-flops, and others.

Xilinx interconnect technology differs from all the previous FPGAs discussed. The interconnections are made by using CMOS transistors as pass gates that either pass or block a signal. They are not permanently wired to do this-it always depends on the logic level applied to the gates of the transistors.

Xilinx and the newer FLEX 8000 family of Altera FPGAs use a static-RAM based technology very similar to the methods of a static-RAM chip. In other words, they use flip-flops to hold information. This information is applied to the gates of the transistors that determine the interconnections as well as the logic itself.

Because it is static-RAM cell based, when the power is removed from the chip, all the interconnection information is lost. As a result, some external device (like an EPROM) must contain this information and the data must be loaded into the chips every time the chip has power applied. This is precisely why they are such good chips for many applications! Because they are not one-time programmable devices, you don't have to throw them away everytime you decide to change the internal logic. On the other hand, the storage chip increases the board space required as well as the chip count.

Xilinx FPGAs have internal logic that can load the necessary information from a standard EPROM or a special 8-pin DIP storage device. It can also be sent this information from another source such as a microprocessor (that may already be in the system) with storage capabilities of its own. This is an application where static-RAM based devices can really shine. By changing a single EPROM, an on-board microcontroller can be updated with new software and the logic in the FPGAs can be changed as well!

Important for experimenters, though, may be that an expensive programmer is not required. Actel and QuickLogic chips require a special programmer to burn the antifuses, and Altera similarly uses an expensive programmer to set up the EPROM-like cells. All that is required for a Xilinx design is a standard EPROM programmer capable of programming 2732s, 2764s or other, similar devices.

SOFTWARE TOOLS

Early programmable logic devices became popular when the software tools made it easy to use them, and FPGAs also rely heavily on the software tools that take a design and generate the programming files.

Each FPGA company has its own development tools, and many FPGA manufactures also work closely with third-party software vendors to provide interfaces with commonly used design entry tools so that implementing a design is as painless as possible. Interfaces are available for OrCAD, Viewlogic, Mentor Graphic, and other popular packages. The human-readable source code for these components is known as a ***Hardware Description Language,*** or HDL.

Besides interfaces, FPGA companies and other vendors usually have tools that allow for entry methods other than schematically. Altera sells a hardware description language (AHDL) that allows for a complete design to be entered and simulated without ever placing a gate. Many companies are providing VHDL options as well. HDLs are a great way to enter a design because the logic is generated for you automatically. The down side is that logic-generating software is expensive.

Until very recently, many thousands of dollars could be spent to buy the vendor-specific software necessary to compile a design, even

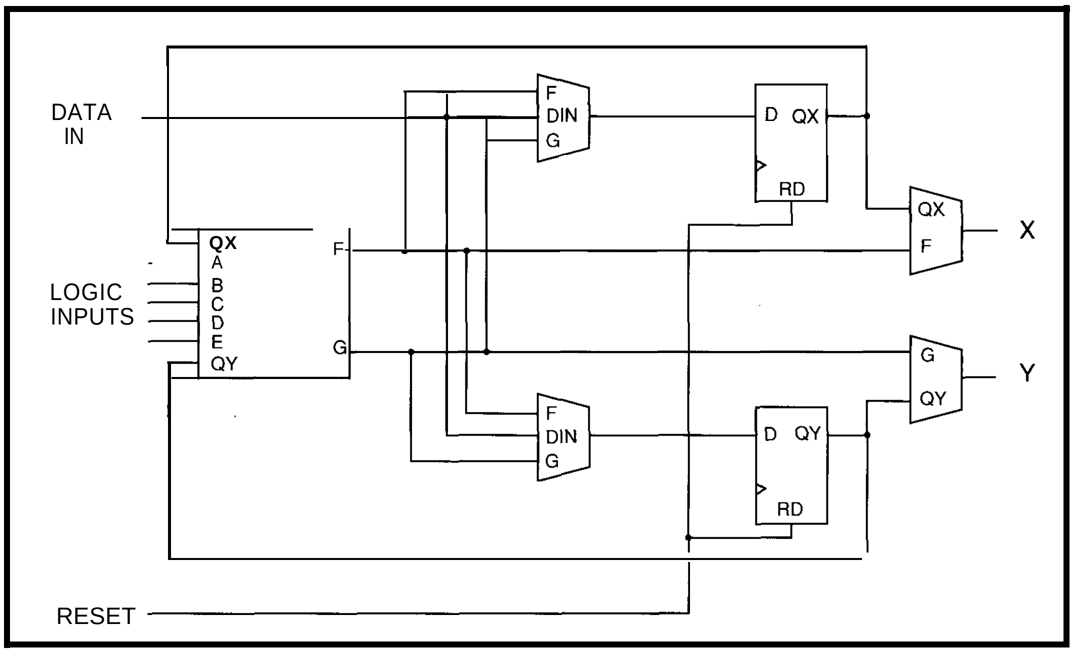


Figure 8—*This is a slightly simplified version of the Xilinx 3000 family CLB. The multiplexers allow for either the combinatorial logic or the flip-flops to be used alone. The flip-flops can be reset and clocked from either local or global signals.*

if the design is initially entered into a third-party schematic-entry program.

Some FPGA vendors such as Xilinx and Actel realized that the software tools need to be less expensive for wider use, so they have come out with basic starter packages for under $1000 that provide all the functionality required but some have limitations on which chips they work with. Even with the limitations, these packages have started to open the market to experimenters and smaller companies to put these useful chips to work.

SEE YA LATER, PAL

Having covered the basics of FPGAs, we are now ready to look seriously at implementing a design. My personal requirements when I was choosing which FPGA included the need for a programmer that I already had available at home (an EPROM programmer), inexpensive software, and an FPGA that would forgive my lack of logic simulation and allow me to reprogram it. Because Xilinx offers some inexpensive software tools and the design can use an EPROM to contain the logic information, I selected a Xilinx 3000 family part.

In the next article, I will go through the complete procedure for implementing a design in an FPGA. The design will be entered into OrCAD and then compiled using Xilinx software to complete the design. The FPGA will be used to create a unique clock with all the logic on one chip!

Del Hatch is an electrical engineer at Sandia National Laboratories. In addition to tracking down project funding, he has developed microprocessor and FPGA applications for control systems.

CONTACT

Actel Corp.
955 E. Arques Ave.
Sunnyvale, CA 94086
(408) 739-1010
Fax: (408) 739-1540

Altera Corp.
2610 Orchard Pkwy.
San Jose, CA 95 134-2020
(408) 894-7000

Crosspoint
5000 Old Ironsides Dr.
Santa Clara, CA 95054
(408) 988-1584

Quick Logic
2933 Bunker Hill La.
Santa CLara, CA 95054
(408) 987-2000

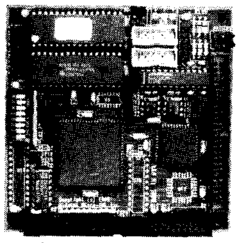
Xilinx Corp.
2100 Logic Dr.
San Jose, CA 95124
(408) 559-7778

IRS

404 Very Useful
405 Moderately Useful
406 Not Useful

We're Small, We're Powerful, And We're Cheaper.

MMT-188 EB
- *2 serial I/O ports*
- *3 programmable parallel I/O ports*
- *1 Meg RAM/ROM capable*
- *powerfail detect interrupt and reset*
- *counter-timers*
- *watch dog timer*
- *expansion connector*



Only $191^{00} (Qty 100)

ALSO AVAILABLE: MMT-Z180, MMT-196, MMT-HC11, MMT-EXP

In fact, you'll get the best product for about half the price. If you're interested in getting the most out of your project, put the most into it. For the least amount of money.

Call us today for complete data sheets, CPU options, prices and availability.

Custom Work Welcome. Call or fax for complete data sheets

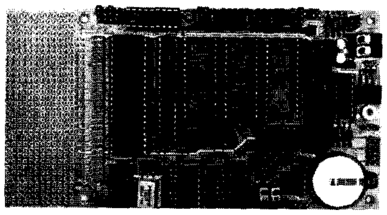
2308 *East Sixth Street*
Brookings, SD 57006
Phone (605) 697-8521
Fax (605) 697-8109

MIDWEST Micro-Tek

WE'RE SMALL BUT WE'RE POWERFUL

#114

8051 EMBEDDED CONTROLLERS
With Lots of Extras!



We offer a full line of low cost 80C32 embedded controllers and software tools which are ideal for developing products, test fixtures and prototypes.

Features Include:
- Low power CMOS design
- Up to 60K of code space and up to 60K of data space
- 5 to 15 volt operation
- Small form factor (3.5" . 6.5") with prototyping area
- System diskette includes application notes
- Start at $100

Available Options:
- Multifunction Board adds A/D, 24 I/O lines and more!
- BASIC-52 or Monitor/Debugger in EPROM,
- C Compiler $100 or BASIC Compiler for $300

Iota Systems, Inc.
POB 8987 . Incline Village, NV 89452
PH: 702-831-6302 . FAX: 702 831-4629

#115

Loader31: A Pseudo EPROM Emulator

FEATURE ARTICLE

Brad Hunting

Tired of having to spend $$ to develop embedded firmware by means of EPROM burning or emulation? Look no further because Loader31 provides the same solution, only at a lower cost.

If you've ever developed embedded firmware, then you are familiar with the code-burn-try-erase cycle of debugging. The algorithms are well thought out and thoroughly tested off-line. The hardware timings are checked against the data books. Now comes the time to put the software and hardware together and see what happens. You burn a ROM and try it. Then you find that subtle defect you missed in testing. That's okay, though, because it's easy to fix the bug, burn another ROM, and try again, and again, and again. After you have been through this loop a few times you start to think "there has to be a better way."

Next you start thinking about ROM emulators. ROM emulators are hardware devices that plug into existing ROM sockets and allow downloading of hex code into the emulator without burning an EPROM. They usually have a serial or parallel connection to a PC host and some software to allow downloading of hex code to the emulator. ROM emulators are available in sizes from a few kilobytes to multiple megabits and can emulate most EPROMs. Prices range from around a hundred dollars at the low end to close to a thousand dollars at the other extreme. Different emulators will provide different added features, including cases, power supplies, the ability to daisy chain—but all provide the same basic ability to download code.

LOADER31

An alternative to an expensive ROM emulator is to use existing target hardware, mix in a little custom software, and duplicate the functionality of a ROM emulator. This article describes how to get the main functionality of a ROM emulator-namely the ability to download hex code-on an 8031 system for only the cost of an SRAM and a socket. I'll present a simple hardware target which implements the necessary memory decoding, and I'll follow up with a discussion of the software that performs the loading of the code under test.

Since this method will not work for all situations I will mention a few of the limitations first. The method of ROM emulation I describe works by mapping RAM space into code space and then downloading hex code to the RAM for execution. The main implication of this method is that the system under test must have at least as much unused RAM address space as the size of the downloaded code. The system must also be able to map RAM space over ROM space and must be able to execute instructions out of the remapped RAM space. This second criterion is easily accomplished on an 8031, but the first constraint is entirely application dependent.

The final constraint for existing systems is that there must be some method of interfacing the new RAM into the system. Therefore, this method may work better for new designs rather than as a solution for existing designs.

The hardware design I presented here implements the memory map shown in Figure 1. The ROM area up

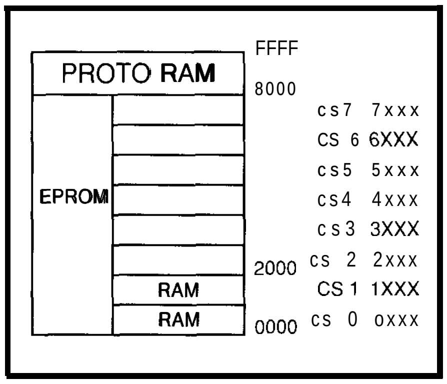
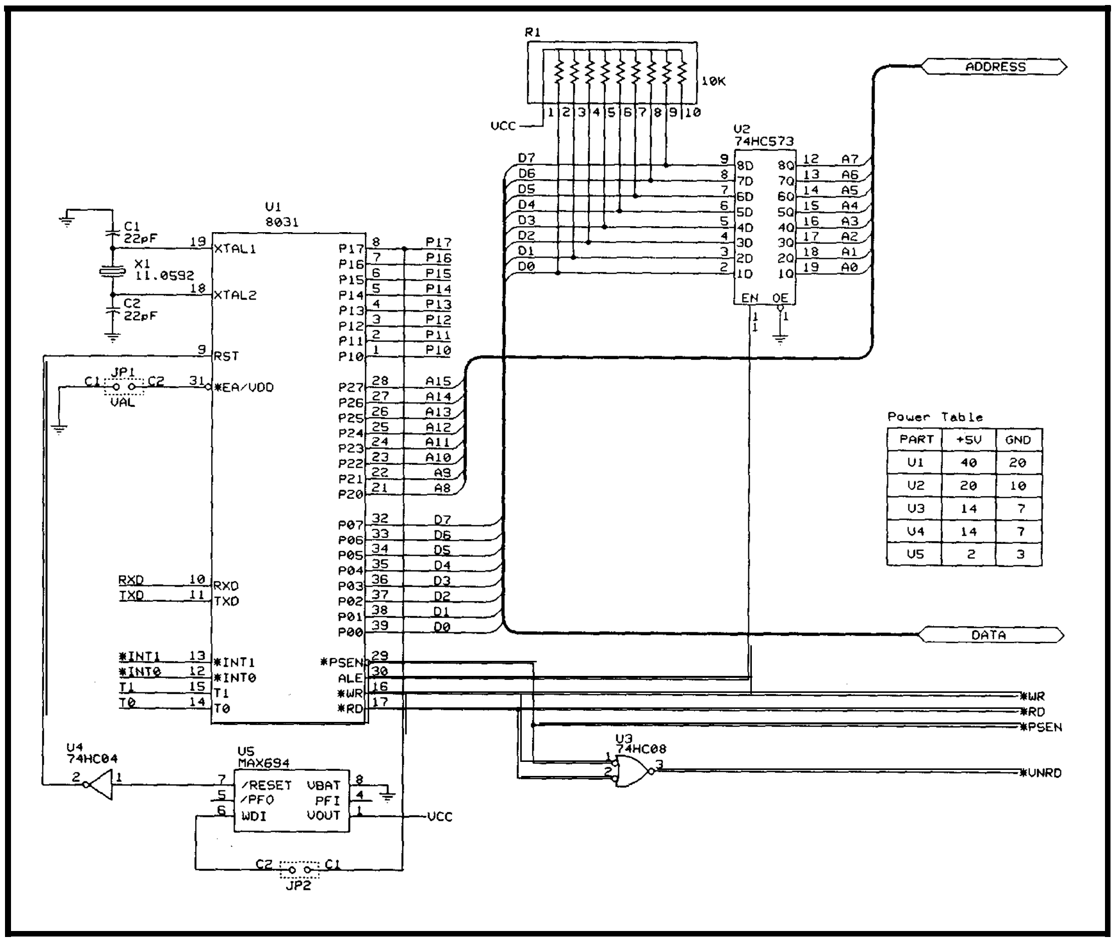


Figure 1--The lower 32K *of code space contains the loader code and any stable application code. The lower* 32K *of data space contains the application* RAM *and I/O decoding. The upper 32K of memory is used for code to be debugged.*



Power Table

PART	+5V	GND
U1	40	20
U2	20	10
U3	14	7
U4	14	7
U5	2	3

Figure **2a**—*Hardware for loader331 consists of an 8031 microcontroller, a MAX694 for power monitoring and watchdog functions, and a handful of glue logic.*

to 32K is used for the system loader and any application code that is known to be stable and defect free. The RAM area up to 8K is used for system and application data space. The RAM space between 8K and 32K is decoded on 2K boundaries for external I/O. The address space above 32K is mapped with ROM and RAM occupying the same space and is used for ROM emulation and code downloading.

HARDWARE

The first thing you'll notice about the hardware when you look at Figure 2 is that there is nothing particularly noticeable. The hardware looks just like the standard 803 1 target that has been presented dozens of times before, which is exactly the idea. This technique is not about any special hardware; it instead shows a way to use familiar hardware more effectively. The only particular item of any interest is the MAX694 used for power monitoring and watchdog timer functions. I am a believer in watchdog circuits for embedded controllers. I have seen controllers lock up in the field and as inconvenient as it is to have a system restart in the middle of a job, it is even more inconvenient to have a system lock up and stay locked up.

A nice feature of the MAX694 is that if the WDI input is left unconnected (JP2 is open), the watchdog timer function is disabled. If the WDI input is pulled high or low and kept in that state for more than 1.5 seconds, then a reset will be generated. This allows the watchdog to be disabled for early development and later reenabled for field deployment. Of course, for low-cost systems, the reset circuit can be replaced by a simple RC circuit.

The target presented in this article has 32K of overlay RAM located at 8000H. Overlay RAM is accessed as either xdata or program space. This is accomplished by ANDing the PSEN and RD lines together. The 8K of data RAM, located at OOOOH, is accessed as xdata only. ROM program space starts at OOOOH and extends for 32K.

SOFTWARE

It's the software that turns this average target hardware into a pseudo ROM emulator. It is called Loader31 and is available for download from the Circuit Cellar BBS. The main routines are written in C for the 805 1. There is one page of startup code written in assembler. The code and MAKE files were written to use the Franklin 805 1 C development tools. The C code should be portable to just about any C compiler.

Pointer implementation is the only area that might cause some problems when porting the code. Given that the 8051 has five different address spaces (data, idata, pdata, xdata, and code), you must take care to be sure that pointers are clearly distinguishable and indicate the type of memory accessed. This is usually accomplished by using three byte-sized pointers. Two of the bytes are used for the address and the third byte contains a number to indicate which memory type the pointer points to.

Loader31 makes extensive use of pointers, so pointer initialization and use should be checked carefully when porting the code. The startup code might need minimal modification depending on the naming convention used by the C compiler for jumping to the main routine. The MAKE file will have to be modified for each individual system to indicate the location of the code and compiler tools.

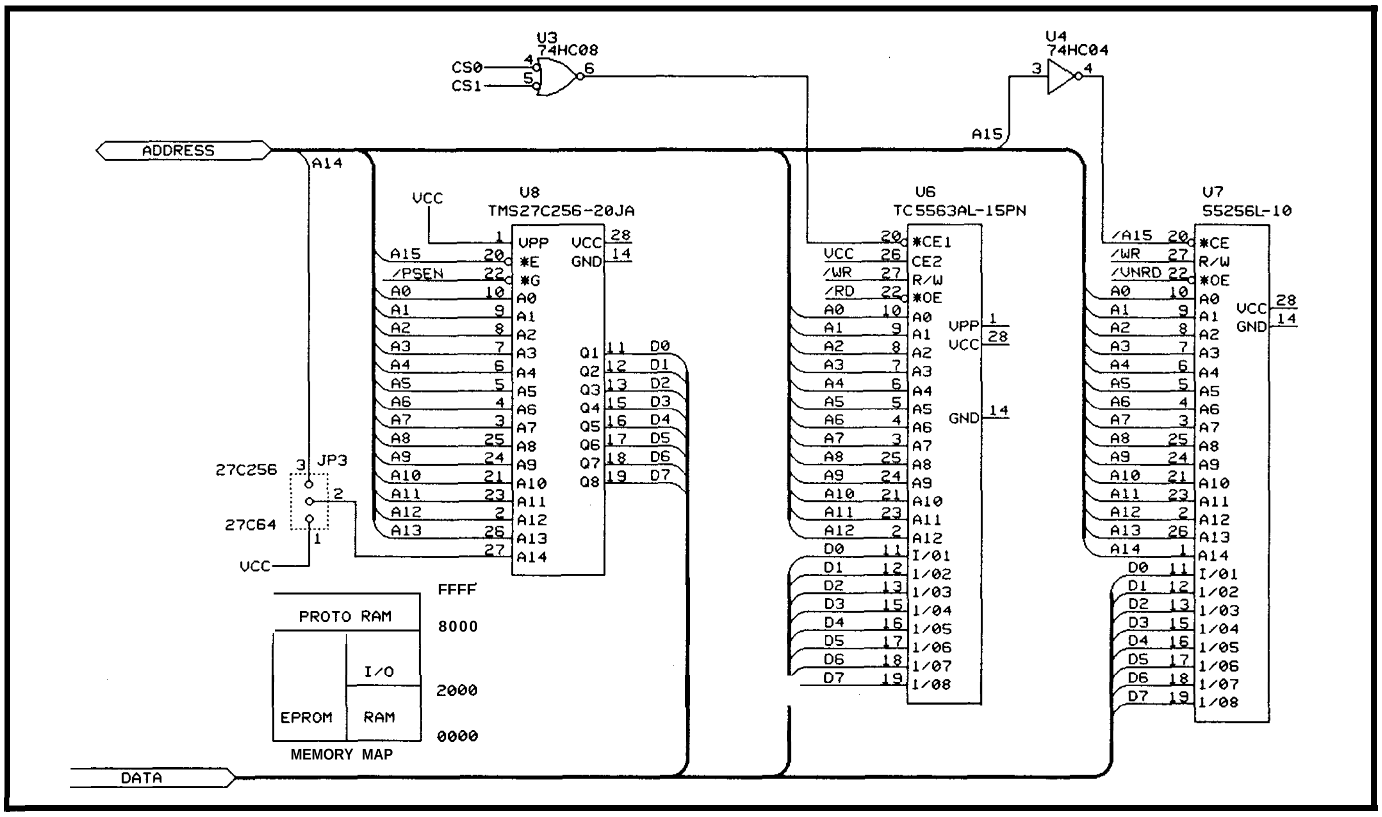


Figure 2b—*The EPROM* contains *the system loader and any stable application code. The small RAM chip contains application storage. Finally, the large RAM chip holds the code to be debugged.*

Loader31 code provides the basic features needed to confirm that a target is up and functioning properly. The loader provides the ability to examine and modify xdata memory, examine and modify the special function registers, examine idata, download Intel hex code, and jump to a specified address. See Figure 3 for a list of menu options. Loader31 occupies a little over 6K of ROM space, so it will fit comfortably in a 2764 EPROM. Loader31 uses the first 256 bytes of the xdata RAM. Application code should only use the RAM space above 1 OOH.

When writing code to be downloaded to the overlay RAM, a few specifics should be kept in mind. First, the code must be relocated to start above 8030H. The overlay RAM starts at 8000H and Loader31 uses the first few bytes of RAM for interrupt redirection.

The second fact to keep in mind is that Loader31 uses the first 100H bytes of xdata RAM. Application code should not use this area. If an error occurs and Loader31 must be reentered, Loader31 will write to the first 100H bytes. If the application variables are located above 100H when Loader31 is reentered, then the application variables can be examined using the (X)dump command.

Third, compiler initialization and startup code must be examined to ensure that there are no conflicts with loading the application code above 0000H.

Fourth, if interrupts are to be handled in higher levels of code, then care must be taken to initialize the memory area between 8000H and 8030H. Loader31 redirects interrupts to this area to enable the application code to capture them. I'll come back to this in a moment.

Loader31 is hardcoded to expect the serial port to run at 9600 bps. This can be changed by modifying values placed in the timerl register near the top of `Loader31.C51`.

Loader31 provides the ability to display and modify the special function registers. Modification of the special function registers is accomplished by using a large `switch` statement with an entry for each SFR. An alternative method of implementing the SFR modification is to write mov and ret instructions out to overlay memory and then perform a call to overlay memory. The disadvantage of this method is that overlay memory must exist. The advantage of using the `switch` statement is that Loader31 can be used to start up a fresh system without any overlay RAM, and SFRs can still be modified.

An area of particular difficulty was getting the jump to xdata address working. As I mentioned earlier, working with pointers for the 805 1 can be a trial. The compiler keeps track of pointer types and will not allow arithmetic operations of pointers with basic types.

INTERRUPT REDIRECTION

The 803 1 is hardwired in silicon to jump to addresses within the first 35 bytes of program space in response to different interrupts. This forces the interrupt handler routines to reside in low memory. In order to allow user applications to handle interrupts, Loader31 redirects the interrupts to overlay RAM by placing a long jump at each of the hardwired interrupt

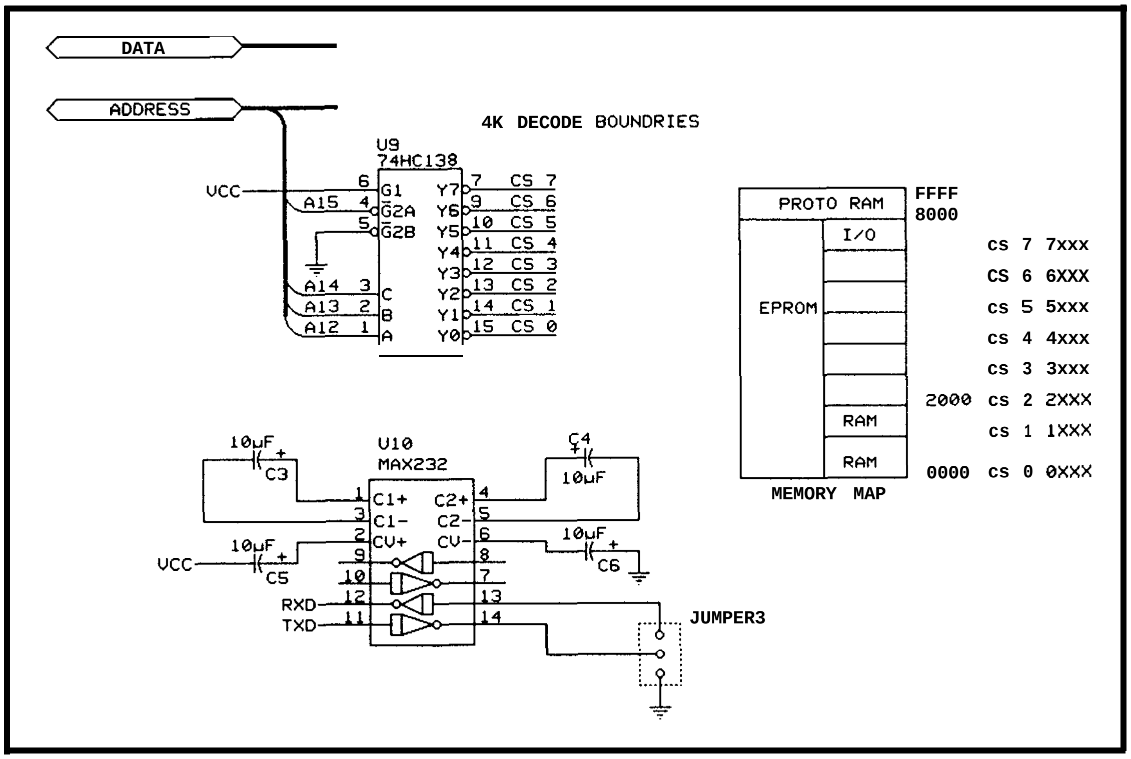


Figure 2c—A simple 74HC138 is used to decode the lower RAM and I/O space, while a MAX232 provides the RS-232 drivers and receivers.

addresses. The loader startup code, STRT_L31.A51, then writes a RET1 at each of the targeted interrupt addresses in overlay RAM. Interrupts are redirected to RAM addresses 8000H through 8020H on eight-byte boundaries. To process the interrupts in the user application code, the user code writes over the RETI instructions in overlay RAM with calls or jumps to interrupt handlers. This can be done either at load or run time.

The example code, STRT_T1.A51, captures the interrupts at load time by placing jump instructions at the redirected interrupt locations. See STRT_L31.A51 or STRT_T1.A51 for exact addresses to modify. Interrupt handlers must end with a RETI or interrupts of equal or lower priority will be blocked. If the interrupt handlers are written in a high-level language, a call to the high-level routine can be placed at the interrupt target address and a RETI placed after the handler returns.

A special note applies to compilers that support the keyword interrupt. If the compiler used to develop the test code allows the interrupt keyword, that feature may not be usable in the Loader31 environment. The compiler may generate additional initialization code when the interrupt keyword is used. This additional code tries to initialize the low memory interrupt vector table which already exists in the Loader31 ROM.

Check the output of the compiler used. If the compiler generates the extra initialization code, then the method of interrupt handling shown in the example test code should be used. See STRT_T1.A51. Tousethis method, the interrupt handler is written in the high-level language as a simple subroutine that neither receives nor returns any values. An LCALL is placed at the target location of the redirected interrupt to call the interrupt processing subroutine. When the subroutine finishes and returns, a RETI is generated to enable interrupts and to return to the interrupted code.

(X)	Dump xdata contents to terminal
(Y)	Dump idata contents to terminal
(M)	Modify xdata location
(F)	Fill xdata with a char
(L)	Load xdata from terminal
(H)	Load an Intel hex file
(J)	Jump to xdata address
(S)	Display SFR registers
(U)	Update SFR registers

Figure 3—Menu options for Loader31 include displaying and updating SFR registers.

CONCLUSION

During the development of Loader31, I ran into some problems with the way the Franklin compiler handles pointers. Luckily, enough of Loader31 was working to allow downloading of the loader code for testing. I have already used Loader31 on a couple of projects and it has certainly helped reduce the development cycle time. ❑

Brad Hunting has industrial experience in embedded systems development for equipment automation and process control. He has recently returned to school to complete a graduate degree at Rensselaer Polytechnic Institute. He can be reached at huntib@rpi.edu or on CompuServe as 7265 7,3456.

SOFTWARE

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

I R S

407 Very Useful
408 Moderately Useful
409 Not Useful

DEPARTMENTS

42 Firmware Furnace

56 From the Bench

62 Silicon Update

68 Embedded Techniques

78 Patent Talk

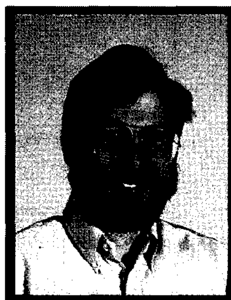
84 ConnecTime

Booting C

FIRMWARE FURNACE

Writing a Micro-C BIOS Extension for the '386SX Project

Ed Nisley



After much cajoling on the BBS, Ed has finally relented and agreed to show how to write a BIOS extension using Micro C. It isn't as hard and doesn't use as much memory as you might think.

"a foolish consistency is the hobgoblin of little minds.. " may be an aphorism suitable for any occasion. In this case, back in Issue 36 I opined that writing BIOS extensions in C wasn't practical because of the Firmware Development Board's limited memory address space. Emerson would smile, as several conversations on the BBS changed my mind.

The consensus seems to be that, if you need just a smidge of code, you may as well write it in C and be done with it. After all, it doesn't matter if you have 3 1 K of C or 3K of carefully tuned assembler.. the RAM still has 32K bytes. I can't argue with that logic, so this month we'll explore the gory details of turning a Micro-C program into a BIOS extension.

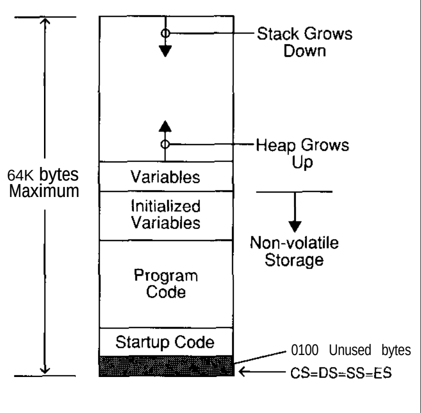


Figure 1--Micro-C's *Tiny memory mode puts all of the program's addressable storage in a single segment that may be up to 64K bytes long. The program code and initialized variables must either be in nonvolatile storage or copied from diskette by a loader program. The* variables, heap, *and stack must be in RAM (for obvious reasons). All four segment registers point to the start of the segment.*

The complexities along the way may seem daunting, but the end result should help you get your own code working.. . and that's what it's all about!

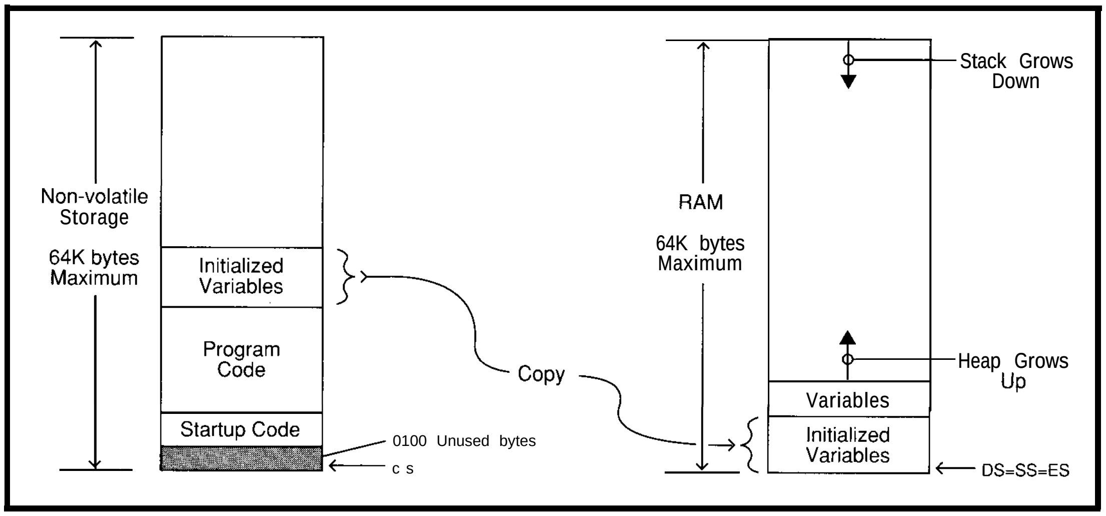


Figure 2—*Micro-C's* Small memory *model allows up to 64K* bytes of *program code and initialized variables, which, as with Tiny mode/, must be in nonvolatile storage addressed by the CS register. The startup code, or loader, must copy the initialized variables into a separate RAM segment addressed by the DS, SS, and ES registers. Because the initialized variables are copied to RAM they may be changed during the program's execution.*

BASIC BIOS BOOTING

Every PC goes through much the same ritual immediately after a hardware reset: check the hardware, find and initialize any BIOS extensions, and finally boot a program from disk. The first time a BIOS extension gets control happens when the BIOS searches for and calls each BIOS extension in sequence. The extensions must hook interrupt vectors if they want to be part of the action after the BIOS regains control.

The exact state of the system isn't predictable when your extension gets control because each extension can add features or change the BIOS setup in nearly any way. In any given system, of course, the same thing happens (or *should* happen!) during each boot, but you cannot assume all systems respond the same way. For example, you'd think the serial ports would be set up before the extensions, but on my '386SX that's not the case.

After all the extensions are set up, the BIOS finishes its own initialization and issues an Int *19* (hex, remember). Under normal circumstances, that interrupt vector points back into the BIOS code to a bootstrap loader routine that handles booting from either diskette or a hard disk. Any BIOS extension, however, can hook Int 19 to regain control just before BIOS accesses the disk; the code may check a serial number, verify a password, or omit the disk boot entirely.

If the original BIOS Int *19 does get* control in a system with no bootable disks, it will issue Int 18 after failing to read the disks. In the Original IBM PC, that interrupt fired up the built-in Cassette BASIC interpreter, but clones don't have rights to that code, so most of them simply display a message and await a three-finger salute. Your BIOS extension can hook Int 18 to get control after the disk boot fails, so you can load one program from diskette or run another from the extension without booting.

According to the references, the system is completely ready for action by the time Int 19 and, if needed, Int 18 occur. Your extension can take control of a perfectly functional PC without having to handle the grubby setup work. The code this month has a "your program goes here" note at the appropriate spot so you can complexicate it as needed.

MODELING MEMORY

So far we have been using Micro-C's Tiny memory model, but that won't work for a BIOS extension. The reason, as with most things PC-ish, involves the segmented memory in 80x86 real-mode programming.

Tiny memory model puts all of the C program's code and data into a single 64K byte segment, as shown in Figure 1. All four of the segment registers must contain the same value: the paragraph address of that segment. An assembly language function or inline code can refer to memory outside that segment by reloading the segment registers as long as they are restored whenever the C-level code has control.

```
a)
:2001000055AA10EB0100BA1C03E...8CC8DA
:200120002D100050B8290150CBB...8B07A6
. . additional lines ...
:2007A000880547EBF620ED7903E...C075AA
:2007C000F75EE81E008A044620C...8805F8
:1D07E00047EBC220ED7901C320C...31
:20000000496E746572727570742...6572AB
:200020000A0020656E746572696...636F1B
    additional lines ...
:200140007573204 96E743139206...2E2E29
:200160000A0020696E766F6B696...2E2E3E
:020180000A0073
:00000001FF

))
 2001000055AA10EB0100BA1C03E...8CC8DA
 200120002D100050B8290150CBB.  8B07A6
    additional lines ...
 2007A000880547EBF620ED7903E.  C075AA
 2007C000F75EE81E008A044620C.  8805F8
 1D07E00047EBC220ED7901C320C.  31
 2007FD00496E746572727570742.  6572A7
 20081D000A0020656E746572696.  636F16
    additional lines
 20093D007573204 96E743139206.  2E2E24
 20095D000A0020696E766F6B696.  2E2E39
 02097D000A006E
 00000001FF
```

Figure 3-a) The *Micro-C compiler produces a single hex file containing both the program code and initialized variables. Because they will ultimately reside in different segments, both sections start at address 0000, but* this *will confuse most hex utility programs and file transfer routines. The code section starts at address 0100 because we are using the DOS COM file format. b) The HEXSEQ utility program reads the hex file and relocates the initialized data addresses so it follows the code section in memory. On/y the addresses and checksums of the last group of lines must change; in this case the data starts at address 07FD.*

Listing la--The *ordinary Small-model Micro-C startup routine assumes it has complete control of the system, so several changes are required to adapt it to use as a BIOS extension. This listing shows the header bytes, the pushbutton test, and the CS adjustment needed to make the code offsets work out correctly.*

```
*
*BIOS Extension header
*
            ORG    $0100          ;standard COM offset
*
            DB     $55            ;signature
            DB     BAA            ; second byte
            DB     16             ;code + init data (512 byte units)
            JMP    <BootEntry     ;jump around checksum
            DB     $00            ; which goes here
*
*- if pushbutton is down, exit without doing much
*
BootEntry   EQU    *
            MOV    DX,#STAT_ADDR
            IN     AX,DX
            TEST   AX,#PUSHBUTTON
            JNZ    <Continue      ;nonzero means not pushed
*
            MOV    AX,#~$0101     ;show to track our path
            MOV    DX,#LED_ADDR
            OUT    DX,AX
*
            RETF                  ; return to normal BIOS boot
*
*- pushbutton is up, so it is OK to continue
*    adjust CS so our offsets are correct
*
Continue    EQU    *
            MOV    AX,#~$0100     ; show single here
            MOV    DX,#LED_ADDR
            OUT    DX,AX
*
            MOV    AX,CS
            SUB    AX,#$0010      ; add 0100 to offsets!
            PUSH   AX             ; simulate a FAR CALL
            MOV    AX,#LdRegs
            PUSH   AX
            RETF                  ; to load the new values
LdRegs      EQU    *
*
```

The program's startup code begins at offset 0100, followed by the compiled C code and library functions. All of the initialized data, such as strings and "constant" variables, come next. Uninitialized variables follow, with the heap beginning just after the last variable. The stack grows downward from the end of the segment, so the heap and stack together use whatever space is left beyond the program and data.

Recall that the first 256 bytes of the segment are a legacy of the DOS COM-format program structure. While this format lets us produce and manipulate the programs with ordinary DOS utilities and compilers, it wastes those 256 bytes. If you have (or write!) the utilities to load the program and set up the registers, Micro-C's startup code can begin at any offset.

A single segment won't suffice for the BIOS extension, however, because the FDB's battery-backed RAM is normally write-protected. Putting variables in write-protected RAM is bad enough, but trying to run a CPU with a write-protected stack just won't work. The assembler code I presented in Issue 38 controlled write protection around each data access and used the caller's stack, but that's impractical for a C program.

The most straightforward solution uses Micro-C's Small memory model, which allows up to 64K bytes of program code and initialized variables in one segment. A second 64K byte segment holds a copy of the initialized variables, the ordinary variables, the heap, and the stack. Figure 2 shows the Small model segment layout.

How this works in our situation should be evident. The code segment is the FDB's battery-backed RAM, while the data segment resides in system RAM below the 640K line. The startup code must reserve the data segment, copy the initialized variables, and load the segment registers before calling the ma i n () C routine. It's just a simple matter of software.. ..

Incidentally, because the startup code copies the initialized variables from nonvolatile storage into RAM, the C program can treat them as ordinary variables with nonzero starting values. In Micro-C's Tiny model, these variables are locked in nonvolatile storage unless you are booting from diskette into RAM, so they behave like constants.

There are other ways to solve the problem, of course. I picked Small model because it worked out quite neatly and exploited an interesting Micro-C feature, but another approach may be more suited for your projects. As always, take what you read here and make your own improvements!

You could duplicate the diskette boot loader's function for the BIOS extension: copy the C program from the FDB's RAM to system RAM, then execute it using Tiny memory model just as before. The battery-backed RAM can thus remain write-protected during the whole operation.

You might also modify the FDB's circuitry to protect the lower 16K bytes of RAM while allowing writes into the upper 16K, but it seems a shame to leave the entire system RAM unused. Micro-C's startup code includes support for such "split" memory segments, which are more common in minimal 8086 systems than PCs. This is, however, a nice solution if you use a PAL to handle memory decoding rather than the FDB's discrete logic.

For now, however, Small model will do what we need without any hardware modifications. There is only one gotcha..

SHUFFLING SEGMENTS

Micro-C was designed as a simple, portable C compiler for microcomputers. Dunfield has adapted it to a wide variety of micros with striking success; it's certainly the best cost-performance buy on the market. The 8086 CPU, however, stretches the compiler's architecture nearly to the breaking point..

The end result of a Micro-C compilation is a hex file in either Intel or Motorola format. In either case, the maximum file size is 64K bytes because the address field is two bytes wide and there is no way to specify more than one segment. Dave is planning support for Intel's extended hex file format, but the trick I'll show you may come in handy in other situations.

The hex file for a Small model program contains two sections: program code and initialized data. The code section starts at address 0100 because we are using the COM file format, but the data section starts at address 0000. The two address ranges overlap without any way to indicate that they are in two separate segments. Figure 3a shows the raw compiler output file.

Because Micro-C was designed for CPUs without segmented memory, there is no way for the compiler to tell the assembler, "This data really has two addresses: one in this segment and another in that segment. Use the right one, please!" The compiler simply produces an assembler file with the code and data at the appropriate addresses in their ultimate segments. That means we must process the hex file produced by the assembler to get what we need.

Rather than tweak the file manually each time,' I wrote HEXSEQ, a DOS utility to modify the hex file's data segment addresses. Because the initialized data should follow immediately after the code in the FDB's battery-backed RAM, the job is quite simple: just add the code's length to each data address. Figure 3b shows what the output file looks like: note

Listing Ib-Because the *Micro-C exfension code is in write-protected, battery-backed RAM, the data segmenf must be located elsewhere. This section of the startup routine reserves 32 kbytes at the fop of the RAM below the infamous 640K barrier by adjusting the BIOS RAM-Size* word *at 0040:0013. It then* copies initialized *variables to RAM, clears the uninitialized variables, and sets up the segment registers.*

```
*locate the data segment at the top 32K of contiguous RAM
* the BIOS RAM size value at 40:13 has units of 1K bytes
*
        MOV    AX,#$057E         ;show r0 on LEDs to mark entry
        CALL   ShowBits
*
        PUSH   DS                ;save BIOS seg regs
        PUSH   ES                ;on the BIOS stack
*
        MOV    AX,#$0040         ;fetch memory size
        MOV    ES,AX
        MOV    BX,#$0013
        SUB    ES:[BX],#>$0020   ;reserve 32K (1K byte units)
        MOV    AX,ES:[BX]        ;convert RAM size to seg reg
        MOV    CL,#6             ;(10 bits left, 4 bits right
        SHL    AX,CL
        MOV    ES,AX
*
* copy the initialized data to the data segment
* the length is just the offset of ?temp...
* as long as ?temp is the first uninitialized variable!
*
        MOV    AX,#$0530         ; r1
        CALL   ShowBits
*
        MOV    AX,CS             ;aim at this segment
```

(continued)

WHY are our cross compilers so inexpensive???

Because we **give** them away **free!**

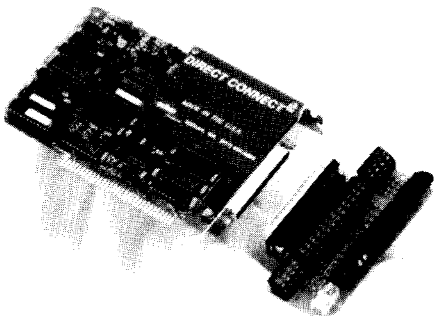
We base our cross compilers on the GNU C/C++ compiler from the Free Software Foundation. We provide you with one year of support*, and give you a ready-to-run cross compiler *with complete source* for DOS, Windows, OS/2 2.0 or UNIX for $495 per year. Or, get the extended support package for $895, which includes GNU *Emacs* and *make,* the CVS and RCS source code control utilities, and the TEX typesetting system. Targets include i386, i860, i960, Motorola 680x0, 683xx and 88000, MIPS and Sparc.

Hundred Acre Consulting
5301 Longley Lane Suite D-144, Reno NV 89511
(800) 2452885 +1 -702-829-9700

#118

PC/XT/AT/386/486 Users!

16 BIT DNA ACQUISITION?



Don't Settle *For* Less
Than The Cutting Edge...

HIGH PERFORMANCE

Guaranteed t 6 bit accuracy
6 Channel A/D Board **$1295**
- 16 bit ND resolution
- 16 bit accuracy
- 50 kHz throughput
- DMA
- 6 lines digital I/O
- 3 channel counter/timer

LOW COST

16 bit 8/16 Channel
A/D Board **$895**
- 16 bit A/D resolution
- 15 kHz throughput
- DMA
- 8 lines digital I/O
- 3 channel counter/timer

OPTIMUM CONVERSION ™

DT2801/5716 compatible
AID Board **$1395**
- 16 bit A/D resolution
- 16 bit accuracy
- DMA, Prog. Gain
- 16 lines digital I/O
- 3 channel counter/timer
- 2 D/A channels

Cut through the specs - each of ADAC's 16 bit boards have been evaluated against every competing model. On noise performance, speed, ease of use, and price, ADAC's leading technology wins every time.

See for yourself - Call for an evaluation board today.

1-800-648-6589

We've been making data acquisition boards for longer than anyone in the world.

adac corporation

70 Tower Office Park, Woburn, MA 01801
FAX (617) 938-6553 TEL (617) 935-6668

Analog & Digital I/O, Industrial PCs, and High Channel Count Systems

#120

Listing 1b-continued

```
        MOV    DS,AX
        MOV    CX,#?temp        ;pick up offset as length
        MOV    SI,#?initdata    ;start of init data in code seg
        MOV    DI,#0            ;copied to our part of data seg
        JCXZ   ?nocopy          ;skip if zero (or 64 K) bytes long
        REP
        MOVSB                   ;from DS:SI to ES:DI
?nocopy   EQU  *
*
* clear the uninitialized data while we have the pointers at hand
* length is the difference between ?heap and ?temp
* as long as ?heap is just beyond the last variable

        MOV    AX,#$056D        ;r2
        CALL   ShowBits
*
        MOV    CX,#?heap
        MOV    AX,#?temp        ;work around for phase error bug
        SUB    CX,AX
        MOV    AL,#0            ;get a zero (or a test value!)
        REP
        STOSB
*
* aim DS at the new data segment
*
        MOV    AX,ES
        MOV    DS,AX

* save the BIOS SS:SP registers
* begin using our stack in the new data segment
* store stack location in interruot vector for the C code
*
        MOV    ?BIOS_SP,SP
        MOV    ?BIOS_SS,SS
*
        MOV    SS,AX
        MOV    SP,#STK_TOP
*
        XOR    BX,BX            ;se  interrupt vector
        MOV    ES,BX
        MOV    BX,#STK_INT
        ADD    BX,BX
        ADD    BX,BX
        MOV    ES:[BX],SP
        MOV    ES:2[BX],SS
        MOV    ES,AX            ;restore ES
```

that only the data addresses and checksums change. HEXSEQ is available on the BBS, but isn't worth further discussion here.

The resulting hex file produces, once again, a simple binary COM-format program that can be loaded into the FDB's battery-backed RAM using the LOADEXT boot loader from Issues 36 and 38. After the code is loaded, BIOS will invoke it each time the system restarts with no further attention on our part.

Now, having the big picture in mind, we can delve into the details of writing a C-language BIOS extension.

SEQUENCED STARTUP

The C startup code must convert the BIOS extension entry conventions into Micro-C's runtime conventions, run the C program, and then return control to the BIOS after the extension is set up. Coupled with the segment shuffling required by Small model code, the startup code has some interesting tricks.

The extension must start with the signature required by the BIOS scan, so Listing 1a is similar to the code presented in Issue 38. The diskette boot loader computes the checksum on the contents of the disk file and the

Listing 1 c-Once *the* variables *and segment registers are set up correctly, the remaining code simply initializes the Micro-C heap and calls the main() function. Unlike most embedded C programs, main() must return so the BIOS can continue the boot sequence. This listing shows how the code restores the segment registers and returns to the BIOS.*

```
*— now, for the main event..
*
        MOV         AX,#$0579          ;r3
        CALL        ShowBits
*
        MOV         <?heap,#0          ;set up empty heap
        CALL        main               ;Execute main program
*
*— restore regs and return
*
        MOV         SS,?BIOS_SS        ;aim back at BIOS stack
        MOV         SP,?BIOS_SP
*
        POP         ES                 ;from BIOS stack
        POP         DS
*
        MOV         AX,#$0505          ;show rr to indicate the end
        CALL        ShowBits
*
        RETF                           ; and continue with boot
*—
*** Remainder of Micro-C startup code is unchanged
```

Listing *2-The main() function shown here moves the Int 19 vector to Int 61 so we can use if later, then aims Int 19 at our own interrupt handler. It displays a tracking number on the parallel port, but cannot send a serial message because the BIOS has yet to identify the hardware ports. Because this function is called during the B/OS extension scan, it must exit to allow the PC to continue booting.*

```
main0  {
     GetVector(0x19,&Int19Seg,&Int19Off); /* move old handler */
     SetVector(0x61,Int19Seg,Int19Off);   /* to new interrupt */

     SetVector(0x19,GetCodeSeg(),INT_S_ENTRY(Int19Handler));

     outp(SYNC_ADDR,0x01);

      return:
}
```

values in the battery-backed RAM beyond the end of the program, so the checksum byte must be zero in the source code and disk file.

The extension bails out without further action if the FDB's push-button switch is closed so you can boot the system without loading the extension. Recall that I rewired the keyboard lock switch in parallel with the push button so you can control this function without opening the case (assuming, of course, that your case is closed in the first place!).

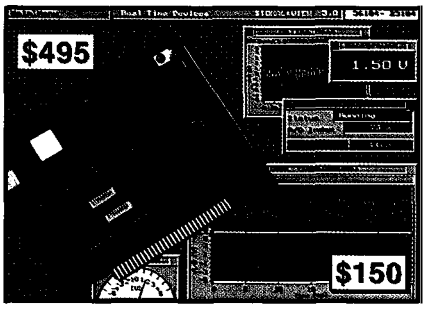
The next step adjusts the CS and IP registers to match the COM program assumption. The FDB's RAM is at C800:0000, so the initial CS:IP is C800:0003. The simulated FAR CA L L reloads CS with C7F0 and adds 0100 to IP so the offsets assumed by the startup code are correct. Refer back to Issue 38 for more details on this trickery.

Next, the code reserves RAM for the data segment. The BIOS power-on self tests record the system's RAM size (in kilobytes) in the word at 0040:0013. This value does not include memory above the 640K-byte line, so a 2-MB '386SX system has a RAM size of only 0280 hex or 640 decimal. If your system has an old CGA card, system RAM can extend up to address

Data Acquisition and Control Without Compromise

Programmable Scan & Burst

RTD's Advanced Industrial Control boards set a new performance standard for general purpose industrial & laboratory applications.



*ADA2210 with SIGNAL*VIEW*
A cost-effective solution

ADA2210 features:

- ■ 125 kHz XT throughput
- ■ 16 SE. / 8 DIFF 12-bit analog inputs
- ■ ±5,±10,0-+10V selectable input range
- ■ programmable auto channel scan
- ■ programmable burst mode
- ■ software & external triggers, pacer clock
- ■ on-demand DMA transfer
- ■ programmable gain: 1/2/4/8 (1/10/100/1 000)
- ■ 3 cascadable 16-bit counters
- ■ 16 programmable digital I/O lines
- ■ two 12-bit analog outputs, selectable range
- ■ rtdLinx Universal TSR DOS driver
- ■ Labtech Notebook driver

HARDWARE SOLUTIONS

Our AIC family also includes AT **analog I/O** boards, **+5V** only A/D & control boards for portable PCs, 4-20 **mA current** loopoutputs, and **opto-22** compatibility.

SOFTWARE SOLUTIONS

Select the power and performance of **SIGNAL*VIEW™** or **SIGNAL*MATH™** from our library of application programs for monitoring, data acquisition and analysis, control, DSP and 3D graphics.

For more information on these and other ISA bus and PC/104 products, call, write or fax us today!

Place your order now and receive SIGNAL*MATH and SIGNAL*VIEW for $195!

rtd®

Real Time Devices, Inc.
P.O. Box *906*
State College, PA 16804
(814) 234-8087 ■ Fax: (814) 234-5218

#121

B800:0000 for 736K bytes of contiguous memory, but that trick won't work with monochrome or the now-standard VGA cards.

The most convenient location for the data segment is at the top of system memory. The code in Listing 1b subtracts 32K bytes from the RAM size, so any subsequent extensions [or the embedded program booted from diskette) will find 608KB of RAM. Admittedly, 32K bytes is more than we need, but you can see the general principle at work.

The next chunk of code copies the initialized variables to RAM. The startup code includes a work variable called ?temp which is always the first uninitialized variable, so the length of the *initialized* block is simply the address of ?temp. If there are no initialized variables, this length will be zero, so I check the CX register before starting the REP MOVSB operation.

The standard Micro-C startup code does not zero the uninitialized variables, but I added that feature because the registers already pointed to the right place. The ?heap variable marks the start of the heap, which always follows the uninitialized variables, so the length is just the difference between ?heap and ?temp. There will always be at least two bytes to clear because ?temp always exists.

The next step is critical. Up to this point, the code has been using whatever stack the BIOS passed to it, but the C program should run with its own stack to provide enough space for nested function calls and interrupts. The code stores the BIOS's SS:SP valuesinthe ?BIOS_SS and ?BIOS_SP variables so they can be restored on exit, then reloads SS:SP with a pointer to the end of the reserved RAM segment.

Figure 2 shows that the stack grows downward in the data segment, so the initial stack pointer aims at the end of the segment. Because I picked a 32K-byte chunk of RAM, the initial SP is 7FFE. If you need a 64K-byte data segment, use FFFE. The value should be two bytes below the end of the segment to avoid the dreaded trap: Stack Fault Error.

Listing 3—*This* macro *wrapper saves the caller's registers, sets up the segment registers for a Small-model C handler, and restores the registers before returning. The code must also save and restore the Micro-C compiler's ?temp work location, but this can cause problems if the interrupted code isn't a Micro-C function.*

```
#define INT_S_PROLOG 63              /* size of prologue code */

#define INT_S_ENTRY(fn) (&fn-INT_S_PROLOG) /* start of prologue */
#define _SPACE_
#define  INT_SMALL(fn,int,stk) asm {\
*- save bystanders on incoming stack \
 fn    PUSH      AX                 \
       PUSH      BX                 \
       PUSH      cx                 \
       PUSH      ES                 \
*- fetch pointer to DS/ES/SS segment and stack          \

       XOR       BX,BX              \
       MOV       ES,BX              \
       MOV       BX,#int*4          \
       MOV       CX,ES:[BX]         \
       SUB       CX,#stk            \
       MOV       AX,SS              \
       MOV       SS,ES:2[BX]        \
       XCHG      SP,CX              \
       PUSH      cx                 \
       PUSH      AX                 \
*- save remaining bystanders and compiler temp on our stack \
       PUSH      DX                 \
       PUSH      SI                 \
       PUSH      DI                 \
       MOV       AX,?temp           \
       PUSH      AX                 \
       PUSH      DS                 \
*- set up seg regs                  \
       MOV       AX,SS              \
       MOV       DS,AX              \
       MOV       ES,AX              \
*- invoke the Micro-C handler       \
       CALL  fn+INT_S_PROLOG        \
*- restore OS, compiler temp. and bystanders from our stack \
       POP       DS                 \
       POP       AX                 \
       MOV       _SPACE_?temp,AX    \
       POP       DI                 \
       POP       SI                 \
       POP       DX                 \
*- restore pointer to incoming stack \
       POP       AX                 \
       POP       cx                 \
       MOV       SS,AX              \
       MOV       SP,CX              \
*- restore remaining bystanders     \
       POP       ES                 \
       POP       cx                 \
       POP       BX                 \
       POP       AX                 \
*- bypass Micro-C return            \
       IRET                         \
}
```

The next step may seem peculiar, but, because our BIOS extension may not be the first or last extension installed, the data segment's location cannot be hardcoded in the startup code. For example, if an earlier extension reserved 10K bytes, our 32K segment would begin at 9580:0000 rather than 9800:OOO0. But we can't store the pointer to the data segment in the data segment.. . .

Fortunately, the interrupt vectors between Int 60 and Int 67 are reserved for user functions. The code stores a pointer to the top of the stack in Int 60's vector. The extension can fetch the Int 60 vector to find its data segment, but actually executing an Int 60 is fatal because the vector does not point to an interrupt handler.

The vector's segment marks the start of the initialized variables at offset 0000. The end of the segment is always two bytes beyond the top of the empty stack, so a single vector can supply three distinct values.

If you're offended by this trick, which has considerable precedent in the BIOS and DOS world, you can put the data segment pointer at a hard-coded location in the FDB's battery-backed RAM. That frees a user interrupt vector, but does require updating the FDB RAM during each boot. Again, it's up to you.

The remainder of the setup routine is almost anticlimactic, as you can see from Listing 1c. The code initializes the heap by writing a zero at ?heap, then calls the main() function. When the routine returns, the code restores the BIOS SS:SP values, pops the saved registers from the BIOS stack, and returns to the BIOS with the obligatory FAR RET instruction.

Unlike all the other Micro-C programs you've seen so far in this column, this main() function *must* exit so the BIOS initialization can continue. Because system setup is not complete, the C program should do no more than absolutely necessary. In particular, on my system the BIOS hasn't loaded the serial port addresses at 0040:0000, so main() can't even send a "hello" message using the BIOS serial functions!

If your system has a video card, monitor, and keyboard, they're ready by the time your extension gets called, so you can display status on the glowing screen and read ordinary keyboard input. I'm taking a minimalist approach to this process, but don't let that discourage you.

If your extension controls hardware similar to the Firmware Development Board's LCD panel, watchdog, or serial number, now is the time to set things up. Don't go overboard; just do the bare minimum and be done with it. You can report error messages on the LED digits, the heartbeat LED, the FDB LCD, or the CRT.

Listing 2 shows a rudimentary main() function that saves the Int 19 Bootstrap-Loader vector and installs a pointer to a customized Int 19 handler. Rather than create a separate storage location for the old vector, the code simply moves it to Int 61. This is also a time-honored PC technique, not a lowdown deceit: it allows you to invoke the old handler with a simple software interrupt rather than a convoluted indirect FAR CALL through a pointer in the data segment. If you must restore the registers before calling the old handler, you'll appreciate why this is a Good Thing.

GETTING THE BOOT

By intercepting Int 19, the BIOS extension will regain control just before the BIOS tries to boot from disk. I covered interrupt handlers in Issue 37, but there are enough differences in Small model to warrant another look at the subject.

Listing 3 shows the macro wrapper for a Small model Micro-C interrupt handler. It saves a few of the caller's registers on the caller's stack, recovers the extension's SS:SP from an interrupt vector location, saves the rest of the caller's registers (including ?temp) on the extension's stack, and then invokes the interrupt handler. When the handler returns, the wrapper undoes all that work and issues an IRET to complete the process.

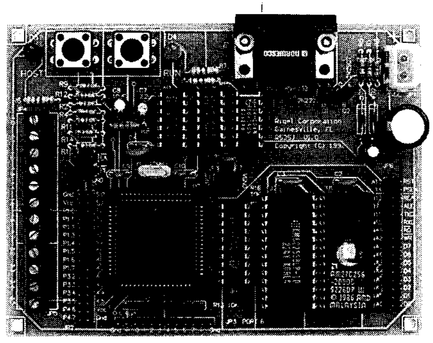
If your code can generate nested interrupts, such as a hardware interrupt during a software interrupt, you *must* place each routine's stack in a different part of the data segment. The third macro argument simplifies this process: it is subtracted from the stack pointer saved in the interrupt vector, so you can allocate separate chunks relative to the original stack top.

I can use the same stack for both Int 19 and Int 18 because my Int 18 handler never returns, but you may need to be more cautious. In particular, make sure you allocate a different stack area for each hardware interrupt

NEW 8031 FAMILY PRODUCTS from RIGEL

FUZZY-LOGIC CONTROL CODE GENERATOR
FLASH (Fuzzy-Logic Applications Software Helper) generates MCS-51 language subroutines to perform fuzzy-logic control tasks from a high-level description of fuzzy rules written with linguistic variables. An extensive tutorial and illustrative examples are provided. ($100.)

R-5355 / READS TRAINER
READS (Rigel's Embedded Applications Development System) and the R-535J board constitute a complete hardware/software development and debugging system in one user-friendly menu-driven environment which runs on an IBM PC host. Programs in the MCS-51 language may be written, edited, assembled, downloaded and debugged without leaving the integrated environment. The R-535J board uses the powerful 80C535 microcontroller. R-535J / READS with User's Guide on disk and example programs is priced at $150, $130 as a kit.



8031 FAMILY MICROCONTROLLERS EXPERIMENTER'S GUIDE
This 300+ page textbook covers the MCS-51 assembly language, using the on-chip facilities of the 8031 family microcontrollers, and software / hardware experiments. Programming nuggets are given for each instruction and each operating mode of the family. Features of the SAB80C535 are highlighted. ($30.)

RIGEL CORPORATION

PO BOX 90040, GAINESVILLE FL, 32607 (904)373-4629

#123

handler and make each stack area large enough that no two stacks ever collide.

Although saving and restoring the registers is routine, the `?temp` compiler variable poses an interesting problem. It must be saved to allow a Micro-C interrupt handler to use that data segment location, but if the interrupted routine is not a Micro-C function, that address is surely not the `?temp` variable! In most cases, saving and restoring a random byte of memory has no effect, but there are some peculiar situations where it can kill your code stone dead.

If `DS:?temp` points to a location that you change as part of the interrupt handler, it will be restored when the handler exits. Should that location be in your stack, well, we pros call that situation "taking a walk in the woods" because a modified return address can send your code nearly anywhere.

Worse things may occur if `DS:?temp` points to a memory-mapped I/O device. These are quite uncommon in PC-land, but the side effects of reading or writing an I/O port can knock the interrupted code off the rails.

This problem is not unique to Micro-C. It's part and parcel of saving and restoring any static location as part of an interrupt handler. One way around this is to verify that DS already points to our data segment before saving `?temp`...if it's aimed elsewhere, don't touch that word!

Lest I leave you with a bad impression, I must mention that the 8086 version of Micro-C makes absolutely no use whatsoever of `?temp`. You can eliminate the (small) danger by simply commenting out the sections of code that save and restore `?temp`. However, future versions of Micro-C may use `?temp`, so the test-before-saving approach is safer. Once more, it's your call.

Your Int 19 handler can take over the world or continue booting as you see fit. Listing 4 shows a simple handler that hooks Int 18, displays a message, and waits for a serial character before continuing. Depending on what you type, it will either invoke the original Int 19 handler or issue an Int 18 directly. If there are no diskettes in the system, the BIOS Int 19 routine will pass control to the customized Int 18 handler.

The sample Int 18 routine displays a message and enters an endless loop updating a counter on the FDB's LED display. Obviously you can be a lot more clever than that. Remember that this is the last chance you'll have to affect the system; I don't know what the BIOS will do if the Int 18 handler returns, as there was no way out of Cassette BASIC!

To summarize, the startup code and `main()` function cooperate to form a BIOS extension that is called during the boot sequence. The `main()` code may hook the Int 19 and Int 18 interrupts to get control before and after the disk boot; those routines are actually software interrupt handlers rather than BIOS extensions, so they should use the macro wrapper to save and restore the registers.

Your code can hook other software or hardware interrupt vectors to

---

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
- HiWIRE II
- EE DESIGNER I
- EE DESIGNER III
- ALL GERBER FORMATS

- FULL TIME MODEM
- GERBER PHOTO PLOTTING

WE CAN NOW WORK FROM YOUR EXISTING ARTWORK BY SCANNING. CALL FOR DETAILS!

Express Circuits

1150 Foster Street • P.O. Box 58
Industrial Park Road
Wilkesboro, NC 28697

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

#124

8051 SBC
AT A NEW
LOW PRICE

We are proud to offer our standard **8031SBC-10** Sinole Board Comouter at a new, low price - just $79 pei unit or as low as $49 each for quantity purchases. An 8031 with two JEDEC sockets, one RS232, 5V regulator, expansion connector. Optional second serial port, **80C31** or 32.

Controller
80C552

At $149, our **552SBC-10** has the price and features you need right now! It's an 8051 core processor with an eight channel, **10-bit A/D**, two PWM outputs, capture/compare registers, one RS232, four JEDEC memory sockets, and more digital **I/O**. And we didn't stop there! You can add options like' two more **RS232/422/485 ports.**, 24 more diojtal I/O ports, **Real-Time** Clock, **EEPROM**, and battery-backup for clock and RAM right on board. Start with the Development board; it has all the peripherals plus a debug monitor for only $349. Download and debug your code -right on the SBC. then move to the OEM board above fdr your production needs. We also do custom design work - call for our reasonable prices.

New 8051 Family
Emulator Support

Our **DryICE** Plus product has been expanded to include support for the Siemens **80C537**. The base emulation unit is still only $299, with the **80C537** pod priced at $199. Other 8051 family processors supported are **8031/32, 80C31/32, 3751152, 87C51/52, 80C154, 80C451, 80C535, 80C552/562, 80C652**, and **80C51FA,B,C**. Each of these pods is priced at $149. Where else can you get an **emulator** with this much power and flexibility for only $448 - complete?

Our original stand-alone 8031 ICE is still **priced** at **$199.** Though not as flexible as the **DryICE** Plus, it offers excellent price/performance for learning or the **occasional** job need.

Call for your custom product needs.
Free Quote - Quick Response

VISA MasterCard AMERICAN EXPRESS Card

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

(619) 566-1892

#125

Listing 4—*The BIOS Int 19 handler normally boots a program from disk, buf this handler gives you a choice. Depending on the serial input character if will either boot using the B/OS handler (which was moved to Int 61 by the startup code) or pass control directly to the Int 18 boot failure handler. You can insert an entire application program either here or in the Int 18 handler!*

```
INT_SMALL(Int19,STK_INT_A,0) Int19Handler() {

int  Option:

  serinit(9600,1);
  enable();    /* allow timer ticks */

  putch(FORMFEED);
  putstr("Firmware Furnace #40 - Ed Nisley\n");
  putstr("Micro-C BIOS Extension Demo\n\n");
  putstr("Interrupt 19 (pre-boot) handler\n");
  outp(SYNC_ADDR,0x02);

  GetVector(0x60,&StkSeg,&StkOff);
  printf(" Int %02x pointer to top of stack is %04x:%04x\n",
     STK_INT,StkSeg,StkOff);

  GetVector(0x18,&Int18Seg,&Int18Off);
  printf(" Capturing Int 18 vector, was %04x:%04x\n",
     Int18Seg,Int18Off);
  SetVector(0x18,GetCodeSeg(),INT_S_ENTRY(Int18Handler));

  while (1){
     putstr(" Press Enter to boot or Esc to skip: ");
     Option = getch();
     putch('\n');
     switch (Option) {
     case '\n':
      printf("invoking previous Int19 handler at %04x:%04x...\n",
         Int19Seg,Int19Off);
asm {
      INT $61

      break:
     case ESC:
      printf(" invoking our Int18 handler...\n");
asm {
      INT $18

      break:
```

replace, modify, or extend standard BIOS services. For example, you could redirect the BIOS Int 10 video routines to use the FDB's character LCD panel.. .it may be a mite cramped, but your applications could use a video display or the LCD with no changes. Get the picture?

RELEASE NOTES

Because BIOS extensions require a modified startup routine that isn't useful for normal C code, I created a separate library file called BIOSEXT. LIB that includes the startup code module 8086RLXS. ASM. You can modify BIOSEXT. LIB toinclude specialized library files or exclude Micro-C library files as needed.

The Micro-C CC86 Command Coordinator has no way to specify a library other than TINY or SMALL, so I've been using my MCCOMP batch file to create the extensions. You can simply replace the 8086RLPS.ASM file with (renamed) 8086RLXS. ASM and use Small model if that is more convenient.

The code on the BBS includes the source files, the demo code, MCCOMP, and the extension boot loader. You'll need the FDB's battery-backed RAM and write-protection circuitry to make use of the loader, although you can use EEPROM with an emulator instead.

I still think 32K bytes isn't enough room for a complete PC program, but this month's sample code should provide a framework for building BIOS extensions and diskless programs. Just don't get carried away and smash into that 32K limit halfway through a project!

Next month I plan to explore Paradigm's LOCATE and TDREM programs. Although Micro-C is a great cost-performer, I know many of you have been eager to use your Borland or Microsoft compilers with the Firmware Development Board.. .stay synched! ❑

Ed Nisley, as Nisley Micro Engineering, makes small computers do amazing things. He's also a member of the Computer Applications Journal's engineering staff. YOU may reach him on CompuServe at 74065,1363 or through the Circuit Cellar BBS.

SOURCES

Dunfield Development Services has just released Micro-C 3.0, which includes structures and unions as well as a "DOS-less Utilities Diskette" that automates some of the hocus-pocus I've described here. Contact:

Dunfield Development Services
P.O. Box 31044
Nepean, Ontario K2B 8S8
Canada
(613) 256-5820
Fax: (613) 256-5821

Pure Unobtainium has the complete Firmware Development Board schematic, as well as selected parts. Write for a catalog:

13109 Old Creedmoor Rd.
Raleigh, NC 27613
Phone/fax: (919) 676-4525

I R S

410 Very Useful
411 Moderately Useful
412 Not Useful

8051 **68HC11** **68HC05** **COP8**

THE INCREDIBLE NEW PE-835 1 FX IN-CIRCUIT EMULATOR ONLY $1451.00

- Supports 8X51/31, 8X52/32, 8X51 FA/FB/FC
- Real-time and Nonintrusive
- 64K Program Memory 64K Data Memory
- 128K Hardware Breakpoints
- 16K Frame Trace Buffer
- Transparent Trace (View Trace and Execute Simultaneously)
- Fast, Easy Installation to RS232 Port of any DOS PC, even Laptops!
- Built-In Self-Test
- Symbolic Debug
- Source-Level Debug

Other Fine **Emulators**
- 8051 From $1400*
- 68HCII From $1600*
- 68HC05 From $1700*
- COP8 From $625*
- More than 100 devices supported through interchangeable probe cards

New Debugger Enhancement8
- Full support for structures, unions, arrays, pointers
- Data structure browser/editor
- True expression in watch window-detect bad pointers

Nobody **Matches** the **Value** of the PE **Family**

PE Product	Max **Freq.**	**MetaLink** list Price*	**NOHAU** List Price*+
8351 FX	16	$1451	$4820
837511752	16	851	4300
8032-24	24	851	4945
8032-42	42	999	7195

*U.S. Retail price. +U.S. Price List dated 3/92, verbal quote 8/93.

METALINK INVENTED LOW-COST EMULATION IN 1984.
Our new AET Emulator architecture (Advanced Emulator Technology, Pat. Pending) provides unmatched value. MetaLink also delivers leading edge customer service, including a 30-dav money back guarantee, 10 dav trial periods, rental plans and free technical support. Call today for FREE demo diskette.

(800) 638-2423 MetaLink® Corporation

MetaLink Corporation
325 E. Elliot Road, Chandler, Az 852'25
Phone: (602) 926-0797 Fax: (602) 926-1198

MetaLink Europe GmbH
Westring 2, 8011<85614> Kirchseeon-Eglharting
Teleton (08091) 2046, Telefax (08091) 2386

Talk on the Phone Without a Word

We've all heard of people who can't walk and talk at the same time. Jeff shows you how to let your fingers do the talking while letting them do the walking. It works even if you are tone deaf....

FROM THE BENCH

Jeff Bachiochi

When we visualize communication without speech, it is natural to think of the hearing and speech impaired who use signing. But sign language is used by many wherever aural communication is impractical.

The steel worker directs the placement of mammoth girders by the simplest of hand movements. His distance from the crane operator requires the use of signing to communicate.

Giant aircraft are shoehorned into crowded terminals with the aid of sign language. Our vocal cords are no match for engine noise and sealed cockpits.

A scuba diver must communicate by gesture because there is something else stuffed in his mouth that gets in the way of talking..

The old saying "a picture is worth a thousand words" may be true in many situations, but in other situations, a word or two will do fine thank you very much. Last month, I introduced the new telephone interface for the HCS II. Adding the speech synthesizer option gives your HCS a voice. This comes in handy if you wish a report on your home's status over the phone. But what if you don't have the voice output option? Most of us can't interpret DTMF tones directly, but what if we could?

BEEP ONCE FOR YES, TWICE FOR NO

If only we could make sense out of those tones. Take a look at your phone's keypad. Notice that in addition to the numerals, the buttons labeled 2-9 also have letters on them (Q and W are missing, though). Today, these alpha characters replace numerals as in the latest ad campaign, for instance: 1-800-COLLECT. When I was younger, our town's exchange was identified by a two-letter prefix to our five-digit local number. So, our phone number was something like TR5-4513. The "TR" stood for "TRemont"-the exchange's ID-and the digits "87."

Although these alpha characters can represent digits, it is a bit more tricky to make digits identify characters. After all, each digit could be any one of three possibilities. If we can assume the "*" and "#" will not be used to represent themselves, we can reassign their use for "alpha pointing."

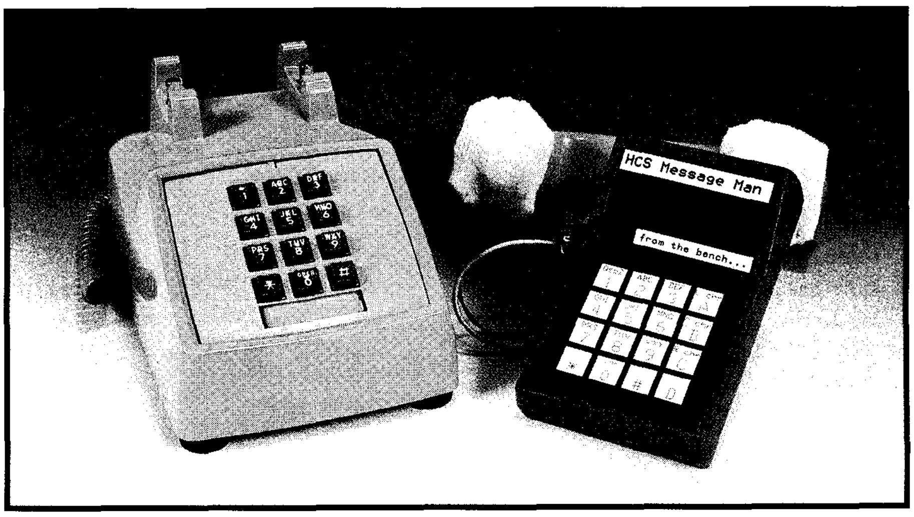


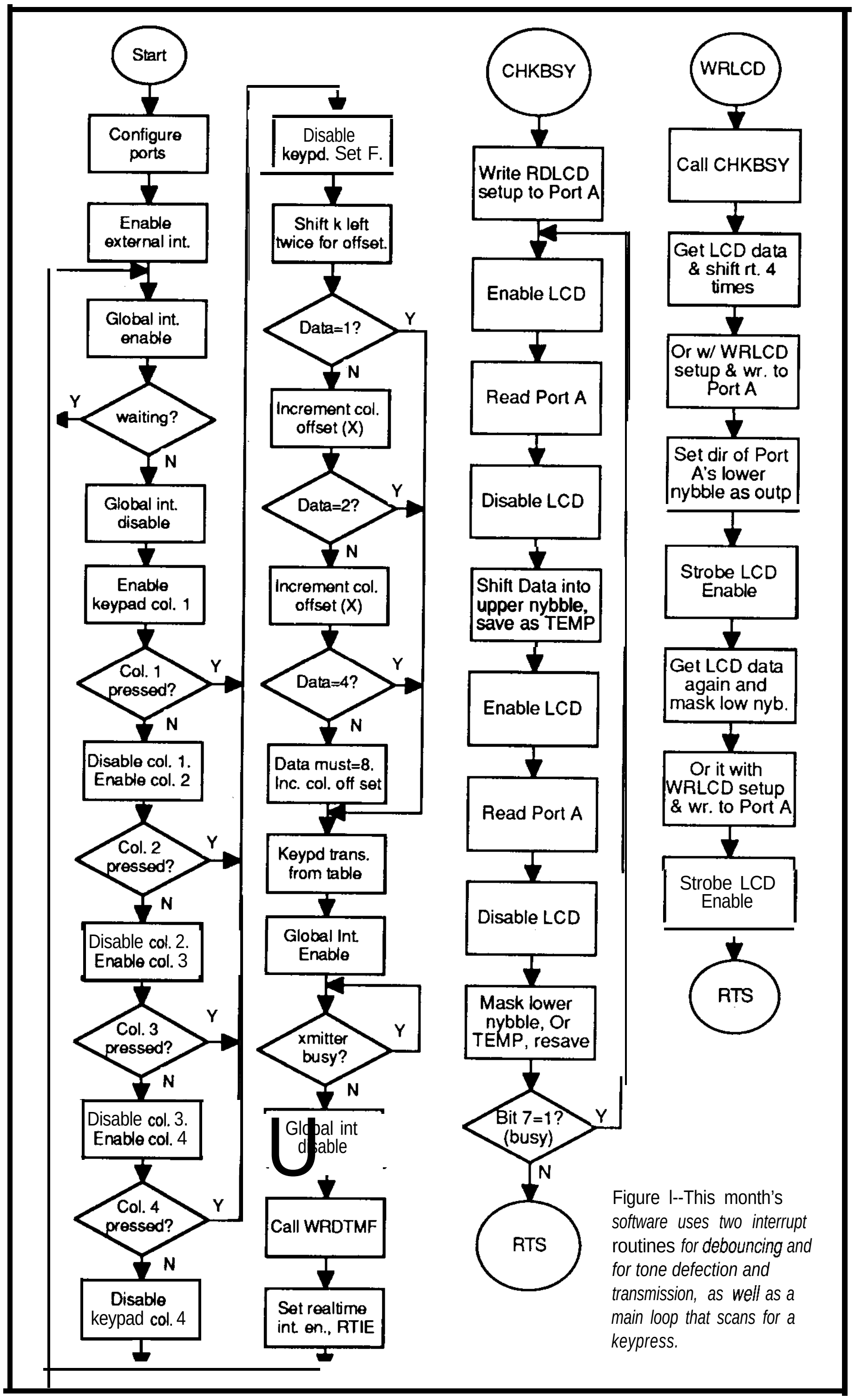
Photo I--The Message Man interfaces with *the telephone through the handset. A microphone and speaker are held onto the* handset with strips of Velcro. The self-contained unit displays messages on a single-line *16-character* LCD *display while it accepts user input on a 16-button keypad.*

That is, any digit preceded by one or both of these characters represents not the digit, but one of that digit's characters. The actual character is based on the pointer.

One possible scenario might be a "*" points to the left character, a "#" points to the right character, and when both of them arrive together they point to the center character. For example, the following meanings could be decoded from these sequences based on the "2" button:

2	=	2
● 2	=	A
● #2	=	B
#*2	=	B
#2	=	c

This method does require up to three tones per character, but it opens up a whole new approach in communications, that is, with the help of this month's project.



Start
Configure ports
Enable external int.
Global int. enable
waiting?
Y
N
Global int. disable
Enable keypad col. 1
Col. 1 pressed?
Y
N
Disable col. 1. Enable col. 2
Col. 2 pressed?
Y
N
Disable col. 2. Enable col. 3
Col. 3 pressed?
Y
N
Disable col. 3. Enable col. 4
Col. 4 pressed?
Y
N
Disable keypad col. 4
Disable keypd. Set F.
Shift k left twice for offset.
Data=1?
Y
N
Increment col. offset (X)
Data=2?
Y
N
Increment col. offset (X)
Data=4?
Y
N
Data must=8. Inc. col. off set
Keypd trans. from table
Global Int. Enable
xmitter busy?
Y
N
Global int disable
Call WRDTMF
Set realtime int. en., RTIE
CHKBSY
Write RDLCD setup to Port A
Enable LCD
Read Port A
Disable LCD
Shift Data into upper nybble, save as TEMP
Enable LCD
Read Port A
Disable LCD
Mask lower nybble, Or TEMP, resave
Bit 7=1? (busy)
Y
N
RTS
WRLCD
Call CHKBSY
Get LCD data & shift rt. 4 times
Or w/ WRLCD setup & wr. to Port A
Set dir of Port A's lower nybble as outp
Strobe LCD Enable
Get LCD data again and mask low nyb.
Or it with WRLCD setup & wr. to Port A
Strobe LCD Enable
RTS

Figure 1--This month's *software uses two interrupt* routines *for debouncing and for tone defection and transmission, as well as a main loop that scans for a keypress.*

I'M ALL EARS

With a bit of computing power, we can decode those blipity-bleeps into real messages and display them on a single-line LCD. For those times when you may run into one of the last of a dying breed-the rotary phone-a keypad and DTMF encoder can be added. The encoder provides access to all sixteen DTMF tones, which you may find adds a bit of security. Remember, the typical Touch-Tone phone only has twelve buttons and, as such, twelve tone combinations.

The system consists of a keypad, an LCD display, a DTMF encoder/ decoder, and a microcontroller. Priority is given to the DTMF decoder to ensure each decoded tone burst is read, converted, and displayed prior to the receipt of the next tone.

While a keypad decoder would have eased the software burden, I wanted to keep the parts cost to an absolute minimum. So, I used a simple 2-to-4 line decoder as a keypad row driver (actually, I used a '238 instead of half of a '239). The rows are sequentially enabled by the processor while the columns are sampled for switch closures. A switch closure is detected when data other than zero is read. The ideal chip here would have been a tristate device, but since the '238 is not, I added diodes to the column lines. This prevents a keypress (which connects the '238's outputs directly to the data bus) from interfering with other operations during the times when the '238 is disabled.

I chose a single-line, 16-character LCD display for its small size and availability. I got this one for less than $10 from Timeline. It even came with EL backlighting (minus the inverter). The MT8888 DTMF decoder uses a nybble-wide data bus, as would the keypad's columns, so using the LCD in the 4-bit mode seemed appropriate.

A 4-bit data bus reduced the number of signals necessary to 10 plus an interrupt. I wanted to avoid external RAM and ROM, so I turned to the 16-pin Motorola 68HC705K1. Al-

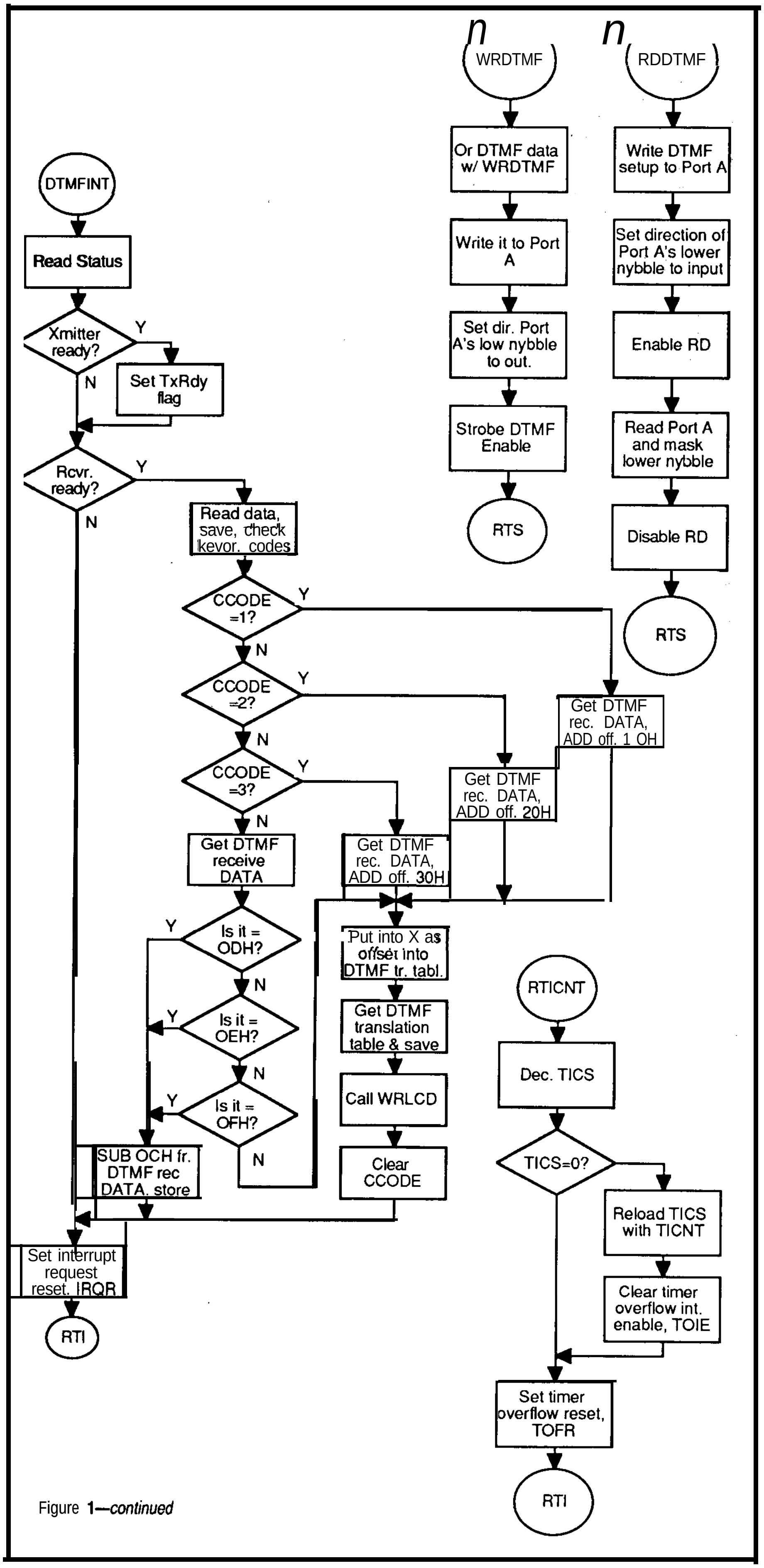

WRDTMF
RDDTMF
Or DTMF data w/ WRDTMF
Write DTMF setup to Port A
Write it to Port A
Set direction of Port A's lower nybble to input
Set dir. Port A's low nybble to out.
Enable RD
Strobe DTMF Enable
Read Port A and mask lower nybble
RTS
Disable RD
RTS
DTMFINT
Read Status
Xmitter ready?
Set TxRdy flag
Rcvr. ready?
Read data, save, check kevor. codes
CCODE =1?
CCODE =2?
CCODE =3?
Get DTMF rec. DATA, ADD off. 1 OH
Get DTMF rec. DATA, ADD off. 20H
Get DTMF rec. DATA, ADD off. 30H
Get DTMF receive DATA
Is it = ODH?
Is it = OEH?
Is it = OFH?
Put into X as offset into DTMF tr. tabl.
Get DTMF translation table & save
Call WRLCD
Clear CCODE
SUB OCH fr. DTMF rec DATA. store
Set interrupt request reset. IRQR
RTI
RTICNT
Dec. TICS
TICS=0?
Reload TICS with TICNT
Clear timer overflow int. enable, TOIE
Set timer overflow reset, TOFR
RTI

Figure 1—*continued*

though the '705K1 is not new technology (really just an '05 core), it competes pin for pin with the same size PIC processor on price, power, and speed.

ANALOG CONNECTION

To make this project universal and portable, and to avoid the expense of a registered DAA (data access arrangement), I made the audio connections through the phone's handset. This eliminates direct connection hassles, but does require you to attach a mic and speaker to the handset during communications. I used Velcro strips to hold these on the handset like a blood pressure cuff around your arm. I used an electret microphone to listen to the DTMF tones because it needs minimum amplification to reach the maximum signal input to the DTMP decoder (1 volt]. The encoder output will not directly drive a speaker, so I used an LM38.6 amplifier to get the necessary power.

Because the phone is designed to allow you to hear your side of the conversation as well as your counterpart's, any button you press that is encoded into a transmitted DTMF tone will be received, decoded, and displayed on the LCD as well. I'm not sure whether to call it a benefit or a shortfall, so let's label it a feature. In fact, it allows the unit to self test.

SOFTWARE OVERVIEW

I used two interrupt routines in this month's project. The first is a real-time interrupt. When using a 4-MHz crystal on this processor, an RTI can be produced every 33 ms. The RTI routine controls a delay period which debounces the switches and prevents the unit from producing multiple DTMF tones. The RTI is enabled whenever a keypress is detected and disabled immediately after the timeout period.

The second interrupt is an external interrupt from the DTMF encoder/decoder. It signifies a tone has been sent or received. If the status register indicates a tone has been sent, a software flag is set to indicate the transmitter is empty. If the status register indicates a DTMF tone has

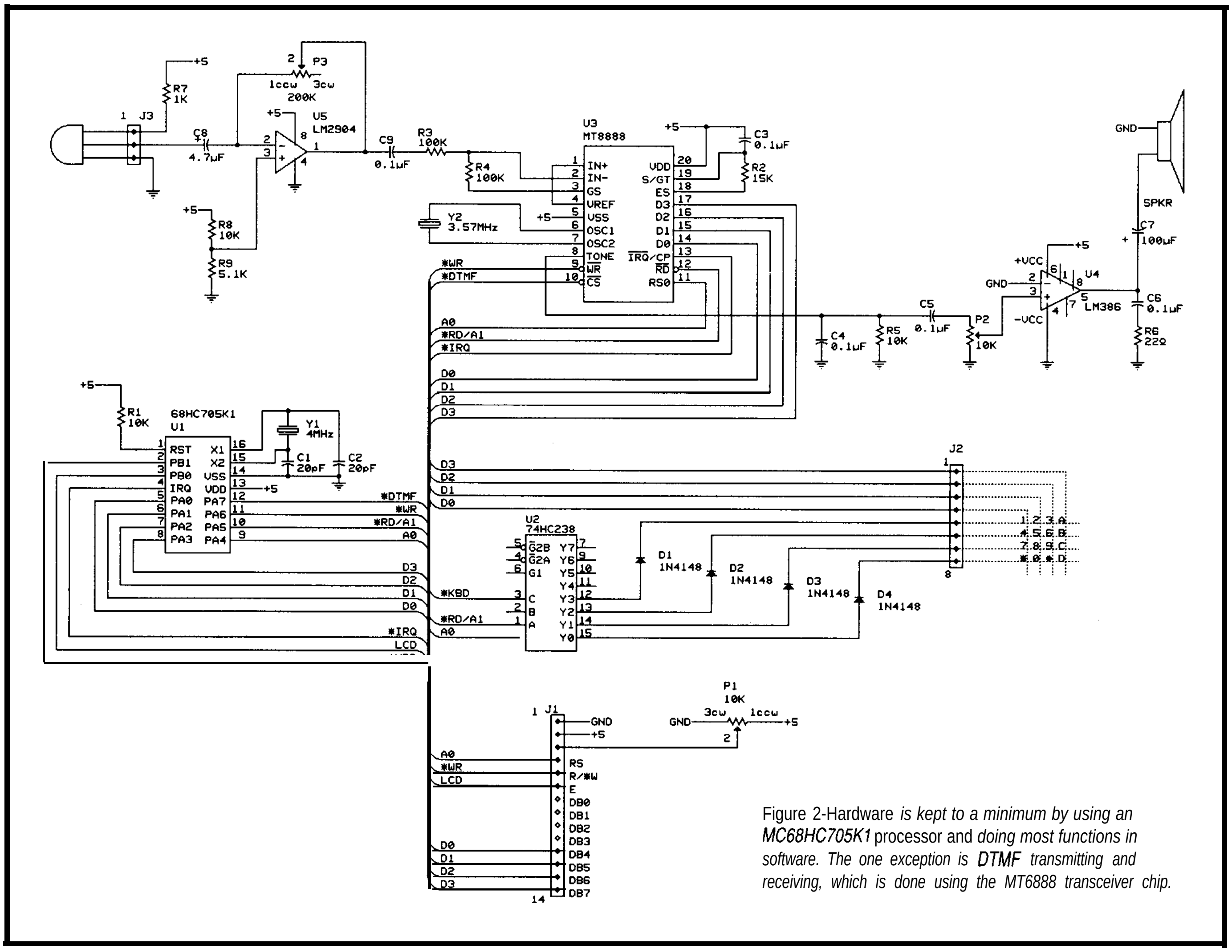


Figure 2-Hardware *is kept to a minimum by using an MC68HC705K1* processor and *doing most functions in software. The one exception is DTMF transmitting and receiving, which is done using the MT6888 transceiver chip.*

been received, the processor reads the DTMF receive register. The data corresponding to the decoded tone pair received is translated into a printable character through a lookup table and sent to the display.

The main loop scans for a keypress. This process is bus intensive and often holds the bus active. If interrupted during a scan, the chances of data being passed successfully are nil (since an active-high output on the '238 will be passed on to the data bus). Therefore, interrupts are globally disabled during these activities.

If the real-time interrupt has timed out and a keypress is detected, the row and column data are used to determine a table offset. This table's data is sent to the DTMF encoder which produces the appropriate tone pair output. Then it's back to look for another keypress. See the flowcharts in Figure 1 for more detail.

4-BIT LCD MODE

The LCD display's 4-bit mode is not difficult to use, but it can be a bit confusing. In 4-bit mode, only the upper 4 bits of data are used, which means all commands must be sent using two 4-bit transfers. The first transfer is the upper nybble and the second is the lower nybble. Simple, right? Well, mostly. You see, the display comes up in 8-bit mode on reset. If the first command sent to it is "use 4-bit mode," then successive transfers will be two 4-bit transfers. But we've only got a 4-bit bus, so how do you send that first command?

This special command uses only the upper nybble (the lower nybble is ignored). The first command is sent as a single 4-bit transfer, and is interpreted as an 8-bit transfer that selects the 4-bit mode. Successive commands are now done with two 4-bit transfers. That goes for reading the display registers as well. Two reads are necessary to receive a full 8 bits.

One other note about the display mode used here. I chose to have the display scroll from the right to left with new characters showing up on the right side of the display, pushing the existing characters one position to the left. This is similar to a moving banner style of sign. This differs from the normal mode in which the cursor moves from left to right.

PACKAGING

If you have a Touch Tone phone and don't need the keypad, DTMF encoder, and audio source, then this project will fit into a very small package. But if you want a 16-button interface or have only rotary dial phones, you will want to build the complete project, which needs a slightly larger container. I used a Pactec HPS-series enclosure.

Listing 1—*Using* the HCS *II's XPRESS* language, conversing with the Message Man is *easier* than you might *think.*

```
! (XPRESS syntax)

! System Config Section

DEFINE Key     = Variable(0)
DEFINE RawTemp = ADC(0)
DEFINE Temp    = Variable(1)

! Program Section

IF Rings>=2 THEN
   OffHook
   Key = DTMFdigit(150)
   IFA Key = 15 THEN                         ! Look for D button
      DialStr("C6B8A8C7C4A3B3B1A8B3A6A7B1"   ! "OUTSIDE TEMP "
      IFA RawTemp < 128 THEN
         DialStr("A0")                       ! "-"
         Temp = RawTemp * 2
      ELSE
         Temp = (RawTemp - 128) * 2
      END
      DialNumber(Temp, 3)                    ! ADC Reading
      DialStr("B0")                          ! Degree symbol
   END
   OnHook
END
```

I made the keypad from discrete push-button switches mounted on a separate piece of protoboard. A single-row, stick-pin connector mates this keypad with a second piece of prototyping material. The circuitry shown in Figure 2 is mounted on the second board.

Graphics for the "Message Man" project cover the LCD display and keypad portions. I printed these on paper and glued them to the back side of translucent mylar to keep the graphics from being rubbed off.

IMPLEMENTATION

The "Total Phone" service offered by our local telephone company uses the "•" and "#" as special function keys to indicate call forwarding and other value-added features. Using these keys as alpha pointers may inadvertently cause calls to be forwarded to somewhere else or cause some other problem. I chose instead to use three of the additional DTMF codes (A, B, and C) as alpha pointers. This improves two things. First, the maximum character sequence length is two ("B5" versus "*#5" to indicate the character "K"). Second, these characters cannot be produced using the standard 12-key Touch Tone phone, so there is a level of security.

Because the "1" and "0" digits do not have any alpha characters assigned to them, the lookup table can assign any characters you want to them [or any other key for that matter). Two alpha characters-Q and Z-are not included on the digits 2-9, so I've arbitrarily assigned them and the space character to the "1" key as follows: Al = Q, B1 = space, and Cl = Z. I took advantage of one of the special characters in the LCD's character generator when assigning characters to the "0" key. This is a small elevated circle used as a degree indicator. The "0" key assignments are as follows: A0 = –, B0 = °, and C0 = ?.

Listing 1 shows what is necessary to send a typical message from the HCS indicating the outside temperature. I assume an 8-bit ADC with a temperature sensor output of O-5 volts indicating temperatures from -250°F to +250°F (0 V = –250°F, 2.5 V = 0°F, and 5 V = 250°F).

If the DTMF code for the "D" key is received by the HCS, then a string of DTMF codes is sent. This string represents the characters "OUTSIDE TEMP." Next, channel 0 of the ADC is compared to a constant (128) to determine if the reading is a negative temperature. If so, the DTMF code "AO" is sent (the minus sign) and variable(1) is set to twice the difference of the reading and the constant. The next statement sends the last three digits of variable(1) (which contains room for four digits and the actual temperature reading). Finally, the string "BO" is sent, which represents the degree symbol.

At your end of the phone, pressing the "D" key tells the HCS that you are ready for a transmission. In turn, the HCS blurts back tones which are displayed as "OUTSIDE TEMP 84°F." Note that this message has more characters than can be displayed on a 16-character display. However, the characters are scrolled in a right-to-left fashion, which allows even rapidly scrolled messages to be easily understood thanks to the persistence of vision.

"...PRESS ONE FOR MORE OPTIONS..."

Now it's your turn to build one of these and receive that coded status report from your HCS without it saying a word. Just don't let the kids see this project or you'll end up building a pair for them. Then again, they may have never heard of Dick Tracy. ❑

Jeff ***Bachiochi (pronounced*** *"BAH-key-* ***AH-key") is an electrical engineer on the Computer Applications*** *Journal's* ***engineering*** *staff.* ***His background includes product design and manufacturing.***

SOFTWARE

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

I R S

413 Very Useful
414 Moderately Useful
415 Not Useful

The Swiss Army Sends in the Special Forces

SILICON UPDATE

Tom Cantrell

Moving on to bigger and better chips, Tom takes a look at WSI's heavyweights in the PLD arena. The PSD4*xx* and PSD5*xx* pack a lot of punch for the money.

You may recall last month's column where I described WSI's PSD3*xx* family of combo chips which combine EPROM, RAM, and glue logic into a single package.

I also described the marketing theory and constraints behind the combo-chip concept. Namely, the niche for such a chip is the window between where single-chip CPUs top out and where the brute-force pricing of commodity chips kicks in. Failure to stay between the lines spells doom, as it did for the circa-'70s first generation of combo chips.

The key point is that, as technology evolves ever onward and upward, the window moves as well. The harsh realities of the ever-changing IC market demand that combo chips adapt-or die.

The PSD3*xx*, like your average GI, does a good job holding ground. It must, because according to WSI, more than 1 million PSD3*xx*s have been shipped to date. However, with further attack on the combo chip niche inevitable, it's time to send the Special Forces forward to prepare for the next assault. Enter the new PSD4*xx* and PSD5*xx*.

BE MY PAL

Fortunately, having explained the PSD3xx in detail last month makes describing these new parts easier. As shown in Figure 1, the PSD4*xx* and '5*xx* (Photo 1) build on their '3*xx* predecessor.

Like the '3*xx*, the new parts combine up to one megabit of EPROM with 16K bits of SRAM. I'm a little surprised that WSI didn't take the opportunity to offer versions of the new chips with more memory than before. After all, some high-end, 8-bit, single-chip CPUs are starting to encroach on entry-level PSD memory density (i.e., 256K bits EPROM). On the other hand, adding memory will be easy-thanks to the on-board bank selection logic-when the need arises. As with the '3*xx* family, versions of the new chips are offered that support

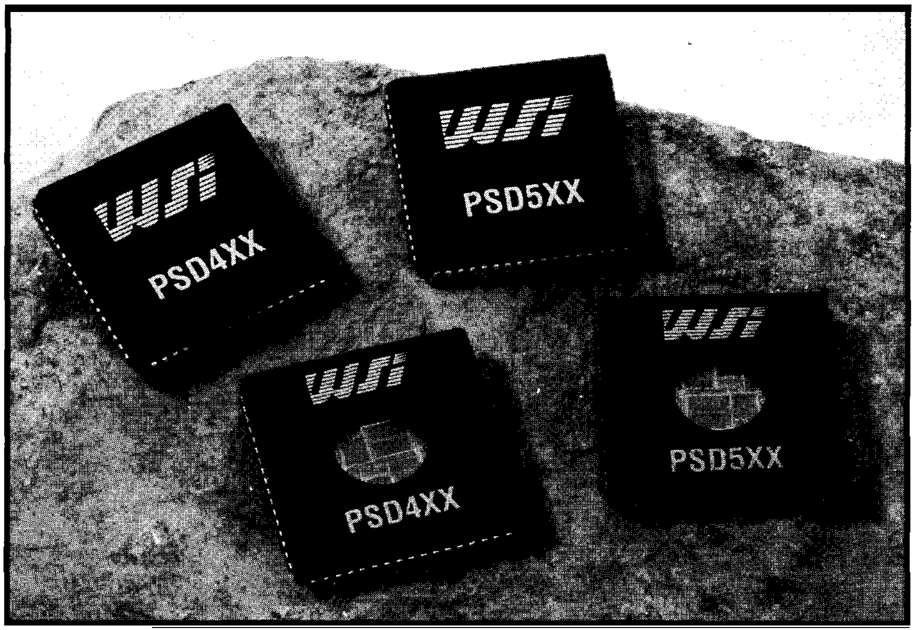


Photo I--A/though *the* new *PSD4xx and PSD5xx series of chips from WSI present the same amount of memory, EPROM access times have left their* predecessors far behind. *An additional 5 bifs to Port C and two 8-bit ports—D and E-have made these chips info a more powerful 68-pin genera/-purpose PLD.*

8-bit ('41x and the '51x) and 16-bit ('40x and the '50x) CPUs.

Though there isn't more memory, what's there is much faster than before, with EPROM access times as fast as 90 ns available. Remember, the PSD access time includes the on-chip decode and MCU interface delay. Thus, an equivalent multichip solution would require a faster-say 70-ns-EPROM to make up for the slowdown imposed by external buffers and decoders.

The major addition common to both the '4xx and '5xx was hinted at in the earlier '3xx parts. You'll remember that a few pins (3 bits of Port C) were available for optional general-purpose use (versus typical use as address inputs or chip select outputs) with the on-chip PADs (Programmable Address Decoders).

Note that what were formerly PAD A and PAD B have now been more clearly named "DPLD" (Decode PLD) and "GPLD" (General-purpose PLD).

The '4xx/'5xx DPLD is little changed from the '3xx PAD A and still serves mainly to map accesses to the EPROM blocks, SRAM, and other internal resources. Indeed, the number of product terms is reduced from 13 to 8, largely due to the switch from an 8-bank to a 4-bank EPROM scheme.

Anyway, the general-purpose PLD functionality offered in the '3xx, while interesting, probably wasn't enough to be of great value in most applications. I imagine WSI was deluged with customers demanding they beef it up.

Since a PLD without I/Os is about as useful as a write-only memory, the first big change with the new parts is the move from a 44-pin to 68-pin leaded (plastic or ceramic, windowed or OTP) chip carrier. Adding five more bits to Port C and two additional 8-bit ports (D and E) boosts, I/O from the 19 lines of the '3xx family to a healthy 40 lines in the '4xx and '5xx families.

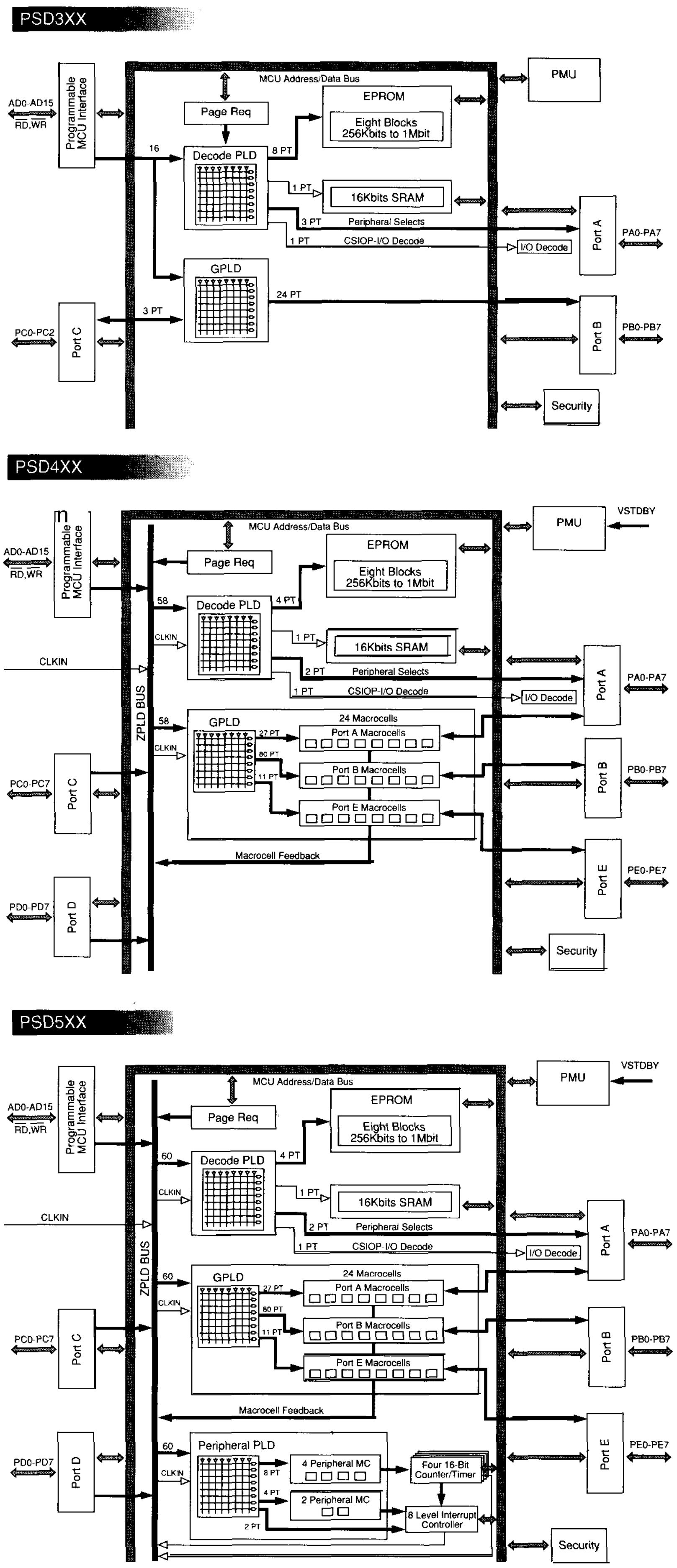


Figure 1--The *5-bit* increase *to* port C as *well* as two additional *8-bit ports* to the '4xx and '5xx families means *I/O lines have extended from 19 (existing on the '3xx* family) to 40 lines.

Having established supply lines, the command post takes the form of a very powerful GPLD (General-purpose PLD). The GPLD offers up to 61 inputs including all 40 port pins plus a variety of MCU bus and internal control signals. One hundred eighteen product term outputs feed three separate groups of 8-registered macrocells (Figure 2) connected to Ports A, B, and E.

In PALspeak, the GPLD is a 61R24 -which is nothing to sneeze at given that a lowly 16R8 costs a buck or so.

SYNCHRONIZE YOUR WATCHES

The '5xx ups the ante even further by adding yet another PLD-the Peripheral PLD or "PPLD"-intertwined with a 4x16-bit timer/counter and 8-level prioritized interrupt controller. Potential interrupt sources include two from the PPLD AND/OR array, two from its macrocell outputs (Figure 3), and terminal count indicators from four on-board timers. Similarly routable, the interrupt controller's outputs can be directed internally to the DPLD and PPLD or via Port E to the CPU.

The clock sources for the timers can be programatically defined as inputs from Port E, outputs of the PPLD, or a new CLKIN pin. The latter features up to 30-MHz input (that's much faster than most timer ICs or timer subsystems on single-chip CPUs) and a programmable divide ratio from x4 to x280.

The timers play a role in improved power management. The original '3xx did have a chip select input that would totally shut down the device to conserve power. Unfortunately, this posed the classic chicken and egg dilemma-how to power up the PSD given that the software for the CPU to do so resides in the PSD! The less-than-ideal solution at the time called for external logic to detect the CPU's first stirrings and splash cold water on the PSD in time for it to deliver the first morning cup of hot opcodes.

Now, thanks to the timers, the '4xx and '5xx can be put in a mode where everything except the I/O ports is shut down automatically after 15 clocks. At this point, power consumption dwindles to a measly 2 μA. Furthermore, the PLDs offer the so-called "zero power" feature in which edge detectors power the PLDs down when the inputs are static. As in the '3xx, a configuration bit can specify that the EPROM be powered down when not selected, reducing power (especially in low-duty-cycle applications) at the expense of slightly lengthened access time.

Another addition on the low-power front is the addition of a new VSTDBY pin which provides dedicated power to the SRAM array. Whenever V_{cc} falls more than 0.7 V below VSTDBY, the SRAM is automatically

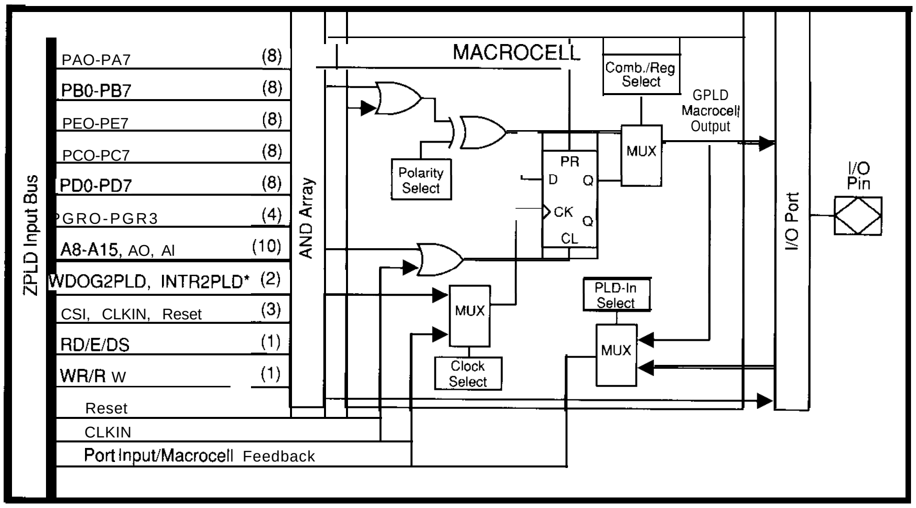


Figure 2—The GPLD offers up to 61 inputs. One hundred eighteen product term outputs feed three separate groups of 8-registered macrocells connected to Ports A, B, and E.

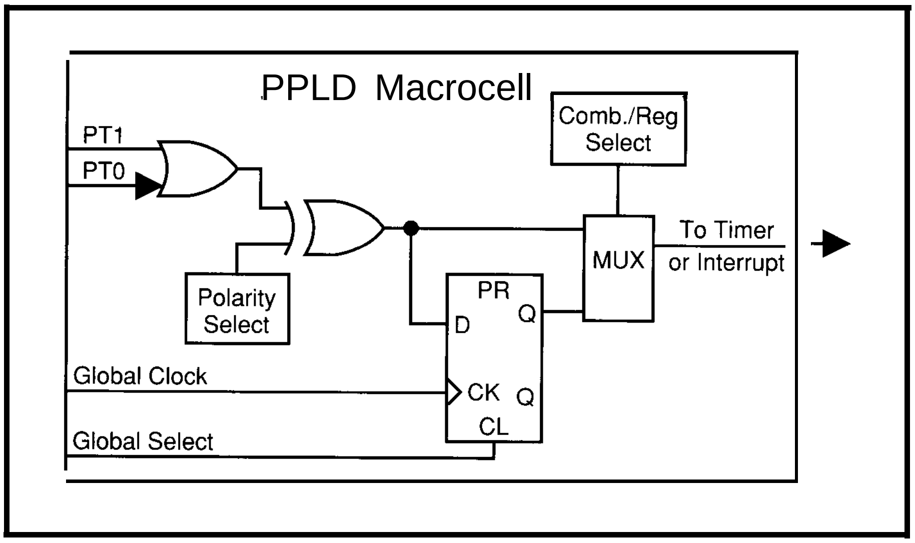
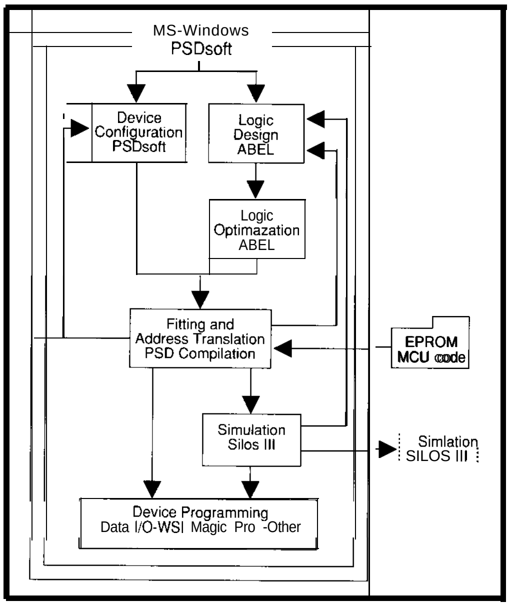


Figure 3-Potential interrupt sources on the '5xx series include two from the PPLD AND/OR array, two from its macrocell outputs (shown here), and terminal count indicators from the four on-board timers.



MS-Windows PSDsoft
Device Configuration PSDsoft
Logic Design ABEL
Logic Optimazation ABEL
Fitting and Address Translation PSD Compilation
EPROM MCU code
Simulation Silos III
Simlation SILOS III
Device Programming Data I/O-WSI Magic Pro -Other

Figure 4-Software for WSI's PSD family is now Windows-based and includes Data I/O's ABEL.

switched into backup mode, consuming minimum power and protected from glitches.

DUELING BUSES

Before we can contemplate the entire spectrum of possibilities posed by the new PSDs, there is one more feature to consider. As shown in the various block diagrams, there are actually two buses on board-the MCU bus (i.e., the typical address, data, and control lines) and the ZPLD bus connecting the D-, G-, and PPLDs.

Note that most resources (such as the ports, timers, etc.) connect to both buses, effectively allowing them to be bridged when it makes sense. However, the appropriate splitting of effort between the buses offers the potential for more optimal designs.

In particular, the ability of the PLDs (and timers and the interrupt controller in the '5xx) to communicate privately allows significant processing to be performed in hardware without host CPU intervention.

Such a setup is ideal for handling a variety of low-level functions such as encoders, decoders, waveform generators, shift registers, BCD counters, and on and on. Benefits include freeing up of MCU bandwidth and interrupts and/or eliminating the need for external chips.

PSD DIVIDEND

Perhaps more than the chips, the software development tool has been radically changed from that offered for the original '3xx. With little general-purpose PLD functionality, the DOS-based MAPLE '3xx software was organized as a kind of fill-in-the-blanks exercise, selecting from a set of predefined functions and options.

The new PSDsoft (Figure 4), besides running under Windows instead of DOS, deals with the general-purpose PLD capabilities of the new parts by incorporating Data I/O's ABEL. Going even further, a Verilog HDL (Hardware Description Language, i.e., C for chipheads) description of the PSD can be simulated with Simucad's SILOS III. Indeed, the Verilog model can be plugged into higher-level board

Free Demo Disk! Call 1-800-221-6630

What is C_thru_ROM?

ROM ***Your*** *Borland* or ***Microsoft C/C++Code.***

C_thru_ROM is the **complete ROM development software tool kii. It lets you run Microsoft and Borland C and C++ programs on an embedded 80x86 CPU without using DO6 or a BIOS.**
C_thru_ROM saves you **money. There are no DOS or BIOS royalties to pay for your embedded systems.**
C_thru_ROM **is complete! It includes the following and much more:**

***Supports Borland's Turbo Debugger.**
***Remote Code View style source level debugger.**
- **ROMable startup code brings CPU up from cold boot,**
- **ROMable library in** source code.

*Flexible **80x86** Locator.

COMPLETE PACKAGE ***ONLY*** $495. ***30-DAY*** MONEY ***BACK*** GUARANTEE.

Datalight®

107 N. OLYMPIC, SUITE 201 • ARLINGTON, WA 98223 • (206) 435-8086 • FAX: (206) 435-0253

#128

AMX™ The Real-Time Multitasking Kernel

680x0, 683xx	80x86/88 real mode
i960® family	80386 protected mode
R3000, LR330x0	Z80, HD64180

Features

- Full-featured, compact ROMable kernel with fast interrupt response
- Preemptive, priority based task scheduler with optional titne slicing
- Mailbox, semaphore, resource, event, list, buffer and memory managers
- Insight"' Debug Tool is available to view systetn internals and gather task execution statistics
- Configuration Builder utility eases system construction
- Supports inexpensive PC-hosted development tools
- Comprehensive, crystal clear documentation
- No-hidden-charges site license
- Source code included
- Reliability field-proven since 1980

*For a **free** Demo Disk and your copy of our excellent AMX product description, contact us today.* Phone: (604) 734-2796 Fax: (604) 734-8114

Count on KADAK.
Setting *real-time standards since* 1978.

KADAK Products Ltd. 206 - 1847 West Broadway, Vancouver, BC, Canada V6J 1Y5

AMX is a trademark of KADAK Products Ltd. All trademarked names are the property of their respective owners.

Multi-Axis

Motion Control

1 to 8 Axis Motor/Motion Controllers

- ***DCX*** controllers incorporate state-of-the-art DSP, RISC; and FPGA technologies to insure maximum performance at minimum cost
- DCX controllers provide many sophisticated ON-BOARD control features such a MULTI-TASKING (the ability to perform independent tasks simultaneously), COMPLEX-CONTOURING, and the ability to programmed directly in USER-UNITS such as inches, meters, etc.
- ***DCX*** controllers can operate totally STAND-ALONE, using their own on-board "computing + memory + I/O" capabilities, or EMBEDDED in a machine. They can also be installed in an ISA-bus (PC/XT/AT), or in a VME-bus (6U)
- DCXcontrollers have a MODULAR ARCHITECTURE for creating a limitless variety of Application Specific MULTI-AXIS controllers. They are created, in minutes, by installing one or more intelligent ***DCX*** *function modules on an* intelligent DCX *motherboard*
- ***DCX*** controllers are supplied with (free) EASY-TO-USE, POWERFUL SOFTWARE for set-up, programming and operating. These include all major HLL interface programs, with examples and source code, a CNC control user-interface with G-code and HPGL capability, and an interactive Servo-Tuning utility

Precision MicroControl
CORPORATION
8122 Engineer Rd. • San Diego, CA 92111

TEL (619) 565-1500
FAX (619) 565-1511

#129

AvCase II 8051 **and** 680x0 Will Travel

Work in the field using your laptop with AvCase II, Cactus Logic's IDS debugger and Promax parallel port EPROM programmer. AvCase II includes an ANSI C compiler, macro assembler, linker, run-time and floating-point libraries with source code, librarian, plus remote monitor and simulator debuggers.

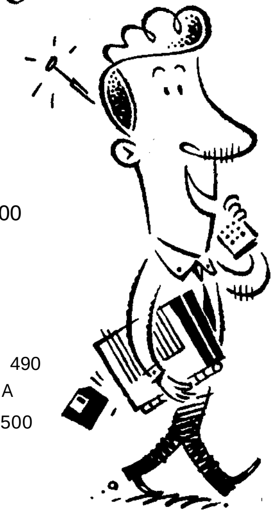
- Integrates all your tools
- Manages the edit/compile/assemble/debug loop
- Compatible with IEEE695 ICEs
- Project management, MAKE, on-line help
- Add your favorite editor, ICE and EPROM

Call for free demo

AvCase **II 8051**
- Supports derivatives from Intel, Philips and Siemens
- New global optimizer produces fast, tight code
- New high-level simulator debugger

AvCase II 680x0 Supports 68000/10/20/30 and CPU300

Save $200 on ***AvCase II until 1 2/31/93***
List Price: $1795 – Your cost $1595



AVOCET SYSTEMS: INC.

120 Union Street, P.O. Box 490
Rockport, Maine 04856 USA
207-236-9055 ■ 800-448-8500
Fax 207-236-6713

Software Development Tools for Embedded Systems

#130

or system models for complete application simulation (don't try it on your '286, though). The price for PSDsoft is $1,495, which isn't as much as it sounds considering it includes ABEL, SILOS III, and the Windows equivalent of MAPLE. Bargain hunters can get a version of PSDsoft without SILOS III for only $695.

Ultimately, as always, the output of all these software machinations is a hex file which can be burned into the PSD with WSI's programmer or some third-party ones, though it may take a while for new '4xx/'5xx adapters to hit the street.

As for pricing, the temptation is to add all the prices of an equivalent multichip setup and tout system cost savings of reduced chip count as the reason to switch. Unfortunately, as more and more features get integrated, the likelihood that a user doesn't need-and thus places no value on—some of the chip's features may increase.

Just introduced, WSI has announced that the '4xx series will start at $6.68 and the '5xx series at $9.81 in production (10k/year) quantities. In my opinion, this pricing is fair given the amount (and speed of] stuff included. Go add it up yourself-fast EPROM and SRAM, high-speed PLDs, 30-MHz timers, an interrupt controller, V_{cc}STDBY switching, and so forth. I think you'll agree. ❑

Tom Cantrell has been an engineer in Silicon Valley for more than ten years working on chip, board, and systems design and marketing. He can be reached at (510) 657-0264 or by fax at(510) 657-5441.

CONTACT

WaferScale Integration, Inc.
47280 Kato Rd.
Fremont, CA 94538
(510) 656-5400
Fax: (510) 657-5916

I R S

416 Very Useful
417 Moderately Useful
418 Not Useful

Decoding Magnetics

EMBEDDED TECHNIQUES

John Dybowski

With the growing proliferation of credit cards with magnetic stripes, it only makes sense that you might want to use one with your next project. It turns out most stripes are encoded in much the same way, and reading them isn't hard.

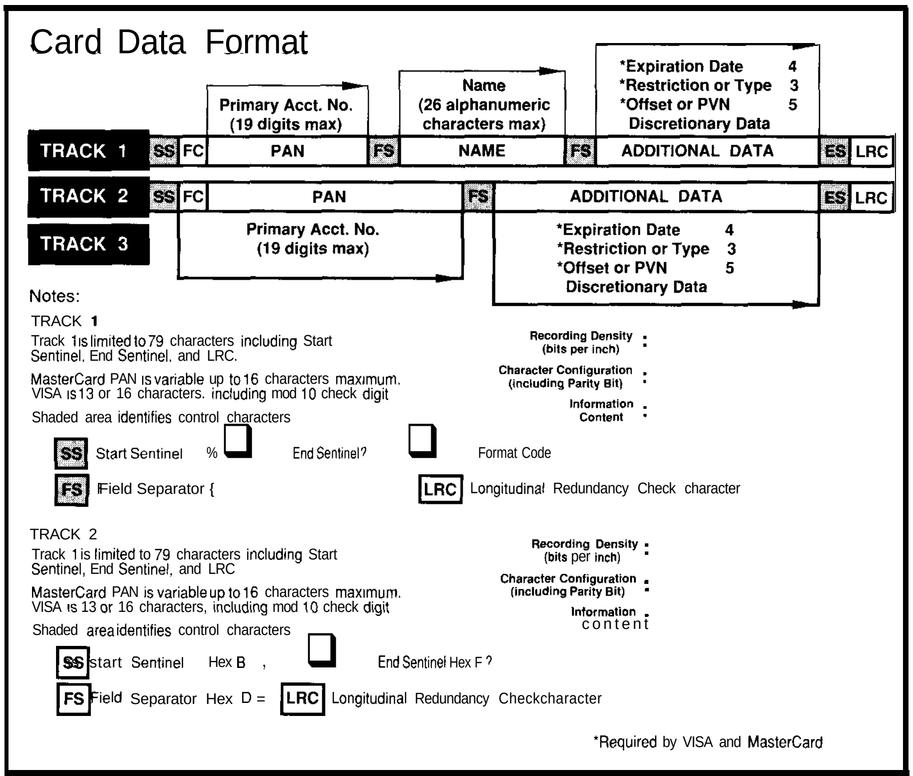
Oast month I wrapped up my dissertation on small memories with a discussion of Dallas Semiconductors' touch memory devices. They are an example of a special type of memory whose utility goes beyond data storage. Possessing the functionality of read-only or read/write-type memories, they cross over into the realm of automatic identification. Automatic identification or data entry addresses concerns such as material and labor tracking, access control, and time and attendance.

These touch memories are the new guys on the block in an industry that has been served well by technologies such as Wiegand wire, bar code, and magnetic cards thus far.

Magnetic media has been used over the years for data storage in such applications as core memories, tape, and disks. Magnetic cards, such as credit cards, should be considered a form of memory exhibiting attributes like those of touch memories. In this regard, this medium not only has the capability of read-only storage, but shares the touch memory's read/write attributes. ..with the right equipment.

MAGNETIC CARDS

Because of their widespread use, most magnetic cards adhere to well-defined standards that describe the physical and magnetic characteristics for a magnetic stripe on a plastic card. These standards outline specifications for a storage format and information interchange. This does not preclude other encoding techniques or additional data tracks for specific applications, but in most cases it makes sense to adhere to at least the basic constraints. This gives you the choice of using any commercially available magnetic encoders in your application.


Card Data Format
Primary Acct. No. (19 digits max)
Name (26 alphanumeric characters max)
*Expiration Date 4
*Restriction or Type 3
*Offset or PVN 5
Discretionary Data
TRACK 1
SS
FC
PAN
FS
NAME
FS
ADDITIONAL DATA
ES
LRC
TRACK 2
SS
FC
PAN
FS
ADDITIONAL DATA
ES
LRC
TRACK 3
Primary Acct. No. (19 digits max)
*Expiration Date 4
*Restriction or Type 3
*Offset or PVN 5
Discretionary Data
Notes:
TRACK 1
Track 1 is limited to 79 characters including Start Sentinel. End Sentinel. and LRC.
MasterCard PAN is variable up to 16 characters maximum. VISA is 13 or 16 characters. including mod 10 check digit
Shaded area identifies control characters
Recording Density (bits per inch) :
Character Configuration (including Parity Bit) :
Information Content :
SS Start Sentinel %
End Sentinel ?
Format Code
FS Field Separator {
LRC Longitudinal Redundancy Check character
TRACK 2
Track 1 is limited to 79 characters including Start Sentinel, End Sentinel, and LRC
MasterCard PAN is variable up to 16 characters maximum. VISA is 13 or 16 characters, including mod 10 check digit
Shaded area identifies control characters
Recording Density (bits per inch) :
Character Configuration (including Parity Bit) :
Information content :
SS Start Sentinel Hex B ,
End Sentinel Hex F ?
FS Field Separator Hex D =
LRC Longitudinal Redundancy Checkcharacter
*Required by VISA and MasterCard

lure l--Most *credit card magnetic stripes are formatted in the same way and include information such as primary account number, name, and expiration date*

The technique used for encoding magnetic cards is known as *Two-Frequency, Coherent Phase Recording.* Allowing for the representation of single-channel, self-clocking serial data, this methodology is generally referred to as *F/2F.* Self-clocking is achieved by combining data and clock bits together in a continuous, synchronous sequence. In this scheme, an intermediate flux transition signifies a one bit and the absence of an intermediate flux transition denotes a zero bit.

Three data tracks defined for use on standard magnetic cards each possess different bit densities and encoded character sets. The average bit density of track 1 is 210 bits per inch (bpi). Track 1 characters are made up of six data bits and an odd parity bit, encoded with the least-significant bit first and the parity bit last, yielding a 64-character set. Taking the number of bits per inch and the number of bits per character, you can see track 1 has the capacity to hold 79 characters.

Track 2 has a bit density of 75 bpi, and track 3 uses 210 bpi. Both of these tracks allow the representation of a numeric-only character set. The characters for tracks 2 and 3 are encoded using a 4-bit binary-coded decimal subset with odd parity and, like track 1, are encoded with the least-significant bit first and parity bit last. The lower density of track 2 allows up to 40 numeric characters, where 107 numerics can be squeezed onto track 3. The actual number of usable characters will be fewer since you also have the Start Sentinel, End Sentinel, and LRC characters.

Though sometimes magnetic cards are moved past the read head mechanically, most applications rely on manually moving the card, either through a slotted reader or into an insertion-type reader. Typically the swipe rate is 5–20 inches per second (ips), with 50 ips being the fastest most card readers can handle. Of course, moving the card by hand will not only result in varying the absolute card velocity but, will also introduce incremental speed changes as the card accelerates and decelerates past the pickup. The F/2F scheme is very forgiving of such speed fluctuations.

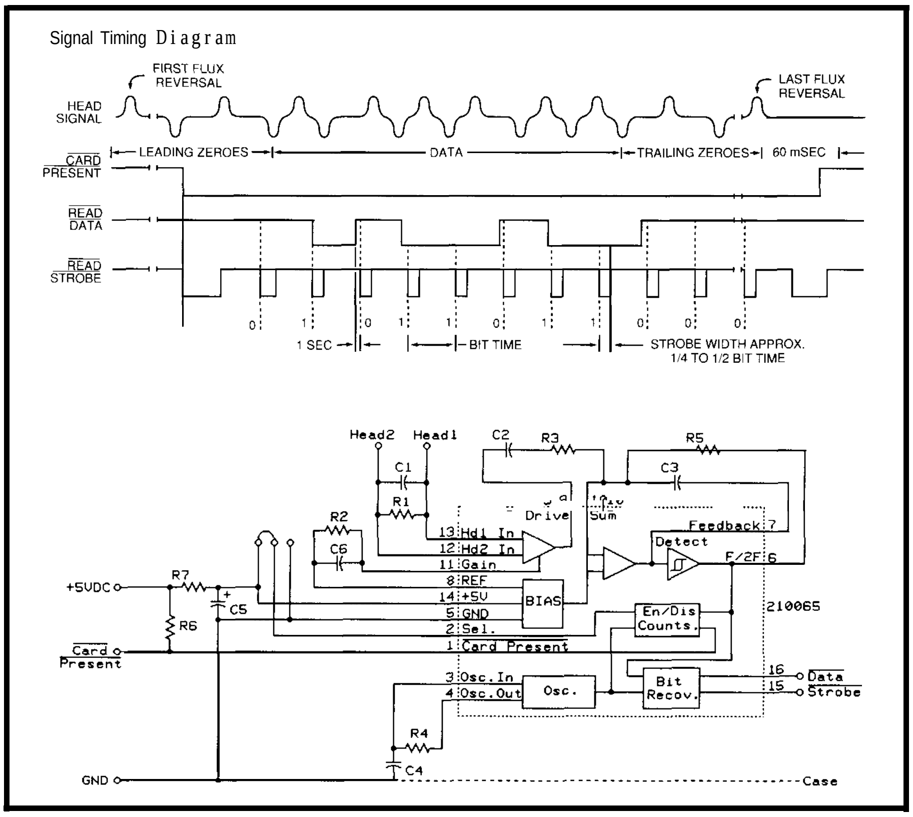


Figure 2—*The* complexities of *conditioning the raw signal from the* **read** *head are typically hidden by single-chip solutions such as the Mag-Tek 21006505.*

Listing 1—*Using* several *externally defined routines, a sample program to read a stripe and store it in a buffer is very short.*

```
            PUBLIC  READ_MAG1
            EXTERN  MS1_BUF (XDATA)     ;sample buffer
            EXTERN  MS1_LIM (NUMBER)    ;sample limit
            EXTERN  MD1_BUF (XDATA)     ;decode buffer
            EXTERN  CP (BIT)            ;card present bit
            EXTERN  M1_CLK (BIT)        ;clock bit
            EXTERN  M1_DQ (BIT)         ;data bit

M1_SS       EQU     5                   ;start sentinel
M1_ES       EQU     1FH                 ;end sentinel

            SEG     CODE
;Sample and decode magnetic track 1
; output: ACC contains character count.
;           DPTR points to data buffer
READ_MAG1   PROC
            CALL    MAG_SAMPLE
            JZ      L?RM1
            CALL    MAGI-DECODE
L?RM1:      RET
            ENDPROC

;General-purpose magnetic sampling routine
; output: ACC contains sample count
MAG_SAMPLE PROC
            MOV     DPTR,#MS1_BUF
            MOV     R1,#0               ;sample counter
L?MS1:      MOV     R0,#8               ;bit counter
L?MS2:      JB      CP,L?MS4
            JB      M1_CLK,L?MS2
            MOV     C,M1_DQ
L?MS3:      JB      CP,L?MS4
```

(continued)

For all three tracks, the fundamental data format is similar and consists of the following elements: First, leading zero bits are encoded to indicate the presence of an encoded magnetic card and provide synchronization pulses to the read head electronics and ultimately to the controller. Next, the Start Sentinel character is encoded which indicates the start of the actual data. The coded data follows. Next, the End Sentinel terminates the data portion of the card and followed by an LRC byte (used for error detection). The LRC is essentially a horizontal parity calculated by an exclusive-OR of all the data bits from the Start Sentinel to the End Sentinel (inclusive). Finally, trailing zeros follow the LRC and fill out the remainder of the card.

Listing 1-continued

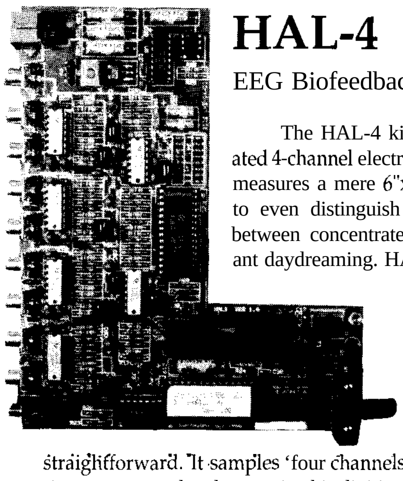
```
        JNB     M1_CLK,L?MS3
        CPL     c
        RRC     A
        DJNZ    R0,L?MS2
        MOVX    @DPTR,A
        INC     DPTR
        INC     R1              ;sample counter
        CJNE    R1,#MS1_LIM,L?MS1
L?MS4:  MOV     A,R1            ;final sample count
        RET
        ENDPROC
```

ANATOMY OF A MAGNETIC CARD

The magnetic tracks have inherent characteristics based on details such as the code set and bit and character densities. International organizations such as Mastercard and VISA impose additional constraints for their participating members, and standards exist for bank debit cards and ATM cards as well. These rules specify the exact content and format of each data field in each track as well the intended uses for the tracks. Naturally, for nonfinancial uses, it is not necessary to comply with these standards. For dedicated uses such as access control, people tracking, and material tracking, adhering to the minimal standards is adequate.

The most-often-used track is track 2, although it offers the lowest information density of the three. It contains all the information that is normally used for credit card transactions. When a customer name is required, track 1 must be used since it the only track that permits alphanumeric data. Track 3 is specified for numeric-only data, but is unique. It is intended for changeable data and consequently may not only be read but may be rewritten by the transaction-handling equipment. A multitude of data fields are contained in these various tracks. Figure 1 shows a brief run-down of what is generally placed on tracks 1 and 2.

CIRCUIT CELLAR KITS



HAL-4

EEG Biofeedback Brainwave Analyzer

The HAL-4 kit is a complete battery-operated 4-channel electroencephalograph (EEG) which measures a mere 6"x7". HAL is sensitive enough to even distinguish different conscious states—between concentrated mental activity and pleasant daydreaming. HAL gathers all relevent alpha, beta, and theta brainwave signals within the range of 4-20 Hz and presents it in a serial digitized format that can be easily recorded or analyzed.

HAL's operation is straightforward. It samples four channels of analog brainwave data 64 times per second and transmits this digitized data serially to a PC at 4800 bps. There, using a Fast Fourier Transform to determine frequency, amplitude, and phase components, the results are graphically displayed in real time for each side of the brain.

HAL-4 kit. ....$179.00 plus shipping

◆ The Circuit Cellar Hemispheric Activation Level detector is presented as an engineering example of the design techniques used in acquiring brainwave signals. This Hemispheric Activation Level detector is not a medically approved device, no medical claims are made for this device, and it should not be used for medical diagnostic purposes Furthermore, safe use requires that HAL be battery operated only!

Sonar Ranging Experimenter's Kit

Targeting ◆ Ranging ◆ Machine Vision

The Circuit Cellar TI01 Ultrasonic Sonar Ranger is based on the sonar ranging circuitry from the Polaroid SX-70 camera system. The TI01 and the original SX-70 have similar performance but the TI01 Sonar Ranger requires far less support circuitry and interface hardware.

The TI01 ranging kit consists of a Polaroid 50-kHz, 300-V electrostatic transducer and ultrasonic ranging electronics board made by Texas Instruments. Sonar Ranger measures ranges of 1.2 inches to 35 feet, has a TTL output when operated on 5V, and easily connects to a parallel printer port.

TI01 Sonar Ranger kit. ....$79.00 plus shipping

CHECK OUT THE NEW CIRCUIT CELLAR
HOME CONTROL SYSTEM

◆ Expandable Network ◆ Trainable IR Interface
◆ Digital and Analog I/O ◆ Remote Displays
◆ X-10 Interface

Call and ask about the *HCS II*

To order the products shown or to receive a catalog, call: (203) 875-2751 or fax: (203) 872-2204

Circuit Cellar Kits • 4 Park Street • Suite 12 • Vernon, CT 06066

#132

REALCARDREADERS

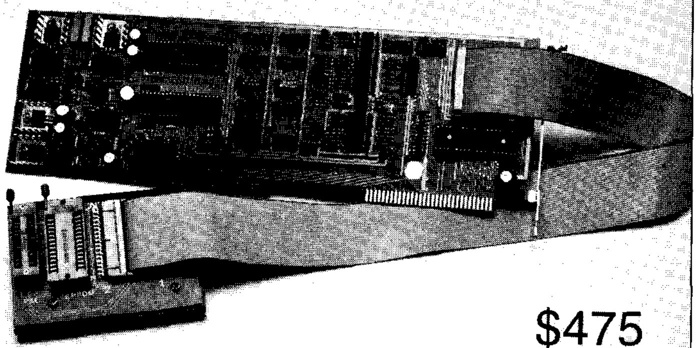
The recovery of magnetically encoded F/2F data can be accomplished directly with the use of just about any microcontroller. There are no particular difficulties in deciphering the raw F/2F data stream and many early magnetic read heads contained nothing more than signal amplifiers and line drivers. These are now artifacts since all modern magnetic read heads contain integrated F/2F bit recovery circuitry and interface with the host controller in a standard fashion using three wires: card present, clock, and data. The read heads usually rely on a single chip to perform the linear signal conditioning, synchronization, and recovery of individual bits from the data stream. The Mag-Tek 21006505 IC is representative of this type of data recovery circuit and its functionality is depicted in Figure 2.

Linear conditioning consists of raising the level of the magnetic pickup's input signal, rejecting common-mode noise, conditioning and detecting the signal, and finally providing a digital output for subsequent processing. The enable/disable counters provide initialization for the recovery section. The recovery section locks onto the data rate and recovers the individual data bits from the data stream. The oscillator section provides the clocks for the recovery section and for the enable/disable timers. Card present goes low after eight or nine flux reversals are seen from the magnetic pickup and will return high about 50 ms after the last flux reversal. The strobe line signals that data is valid and is active low. The Data pin indicates a one bit when it is low. Raw F/2F data can also be picked directly off the chip. Photo 1 shows a typical dual-track magnetic pickup and its associated data recovery circuit card.

The data rate for a high-density track scanned at 50 ips comes to 10,500 bits per second (bps). This results in a transfer rate of 1,500 characters per second for the 7-bit elements used on track 1, and 2,100 characters per second using the 5-bit elements of track 3. In either case, this translates to a new bit arriving at the controller just under every 100 µs.

UNIVERSAL PROGRAMMER

PAL
GAL
EPROM
EEPROM
FLASH
MICRO
87C51
PIC
93C46
XC1 736
PSD 3xx
5ns PALs

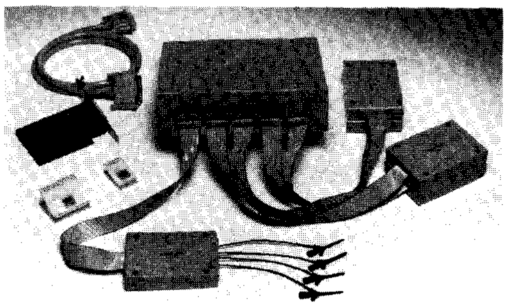


$475

Free software updates on BBS
Powerful menu driven software

400 MHz LOGIC ANALYZER

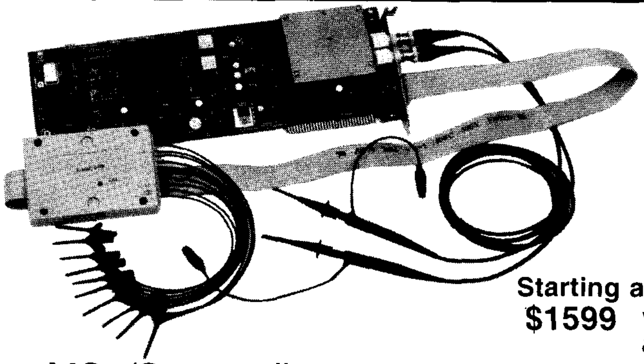
- up to 128 Channels
- up to 400 MHz
- 16K Samples/Channel
- Variable Threshold Levels
- 8 External Clocks
- 16 Level Triggering
- Pattern Generator Option



$799 -	LA12100 (100 MHz,24 Ch)
$1299 -	LA32200 (200 MHz,32 Ch)
$1899 -	LA32400 (400 MHz,32 Ch)
$2750 -	LA64400 (400 MHz,64 Ch)

Price is Complete Pods and Software included

200 MSa/S Digital Oscilloscope



Starting at
$1599 With Pods & Software

- 200 MSa/S Sampling Rate
- 2 Analog Channels (2ch. Digital Osciloscope)
- 8 Digital Channels (8ch. Logic Analyzer)
- 125 MHz Single Shot Bandwidth
- 4K Samples/Channel (Analog & Digital)

Call (201) 808-8990

Link Computer Graphics, Inc.
369 Passaic Ave, Suite 100, Fairfield, NJ 07004 fax: 808-8786

#133

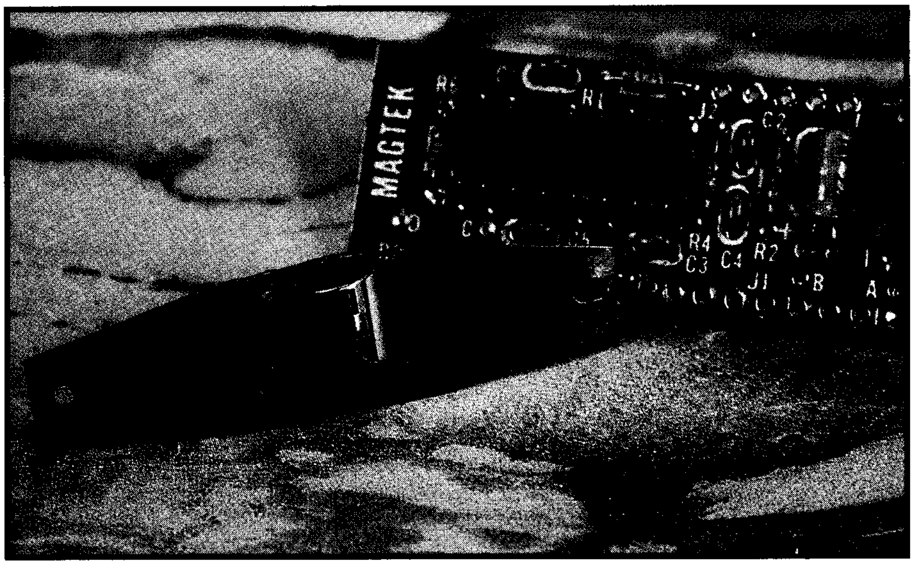


Photo 1—*The compactness of the data recovery circuit board and dual-track magnetic pick up make it possible to build a small identification module for use as a card reader for entering your house or garage.*

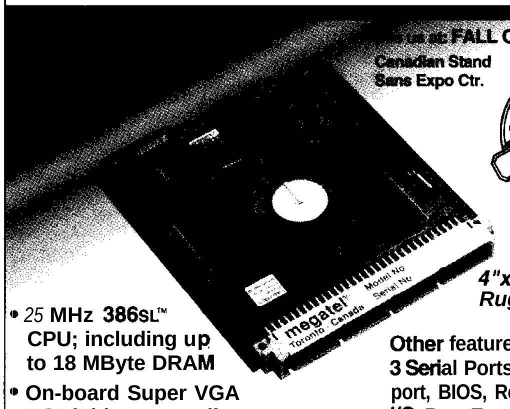
Even the most anemic controller should be able to keep up. With reasonably good coding techniques, there should be no problem handling the entire data sampling phase on an interrupt-driven basis. The low-density (track 2) data flows at a more pedestrian 3,750 bps, yielding 750 5-bit characters per second, or a new bit every 266 us. Since most dual-track read heads provide track 1 and track 2 data, this indicates that handling both tracks simultaneously is feasible under interrupt control. Keep in mind that 50 ips is a rather fast scan rate; 2030 ips is probably a more realistic limit.

MAGNETIC BIT STORMS

When approaching a problem such as decoding magnetic cards, it pays to spend some time looking at the overall picture before starting to write the code. At first glance, it would seem that organizing the data into the prevailing element size during the sampling interval would make decoding easier. This could be easily done by ignoring all the leading zeros, with actual data storage commencing with the first one bit. Of course, this approach assumes you're getting good data. The fact that the data recovery is handled using well-proven hardware makes this assumption valid.

If all you need to do is decode the card in a forward direction, then going about things as I just described makes sense and reduces the coding effort to a trivial exercise. If you have to support reverse decoding, then this is not the optimal solution. Having considered the tradeoffs of being able to decode a magnetic card in both forward and reverse directions, I decided to structure the program to work equally well

Embedded PC
with on-board Ethernet
and Super VGA



[illegible] FALL COMDEX
Canadian Stand
Sans Expo Ctr.

EMBEDDED-PC PC/104 MODULES

4"x 4" Small Rugged Format

- 25 MHz 386SL™ CPU; including up to 18 MByte DRAM
- On-board Super VGA LCD/Video controller
- On-board Ethernet, Featuring AUI and 10 BASE-T interfaces
- On-board SCSI, Floppy, controllers and 2 MByte Flash Eprom Solid State Diik
- +5 volt only operation, 5 watts power consumption

Other features include:
3 Serial Ports, Parallel/Printer port, BIOS, Real Time Clock, I/O Bus Expansion
For more information call:
Megatel Computer Corporation
125 Wendell Ave., Weston, Ont.
M9N 3K9 Fax: (416) 2456505
(416) 245-3324
Megatel is a registered trademark of the Megatel Computer Corp
386SL is a trademark of Intel Corporation

megatel®

#134

Listing 2-Once the serial *bit* stream has *been acquired, the* **decoding** *algorithm takes over.*

```
;Bidirectional magnetic track 1 decode routine
;input: sample count in ACC, output: ACC contains character count
       DPTR points to decoded data buffer
;Reg. usage for this routine (includes subroutines) as follows:
;R5: Direction flag, 0=forward
;R4: Cumulative LRC register
;R3: Decoded character counter
;R2: Nondecrementing sample count
;R1: Decrementing sample count
;R0: Bit synchronizing counter
MAG1_DECODE  PROC
             ;1st pass, setup for forward decode attempt
             MOV    R5,#0           ;indicate forward decode
             MOV    DPTR,#MS1_BUF   ;point to start of buffer
             MOV    R1,A            ;sample counter
             MOV    R2,A            ;sample count
             JMP    L?M1D1
L?M1D0:      ;2nd pass. setup for reverse decode attempt
             MOV    R5,#1           ;indicate reverse decode
             MOV    DPTR,#MS1_BUF
             MOV    A,R2            ;sample count
             DEC    A
             ADD    A,DPL
             MOV    DPL,A
             CLR    A
             ADDC   A,DPH
             MOV    DPH,A           ;point to end of buffer
             MOV    R1,2            ;sample counter
L?M1D1:                             ;decode initialization
             MOV    R3,#0           ;character counter
```

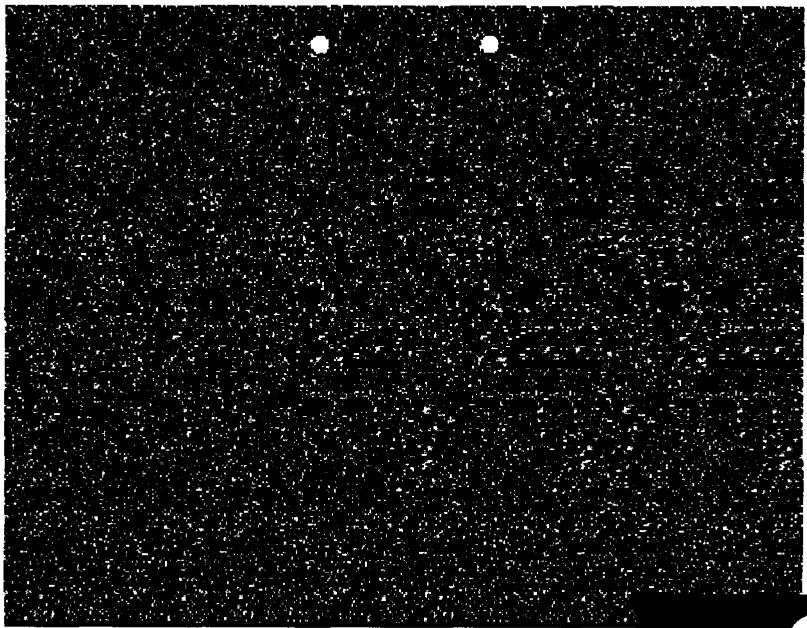
(continued)

in either direction at the cost of a slight increase in initial complexity.

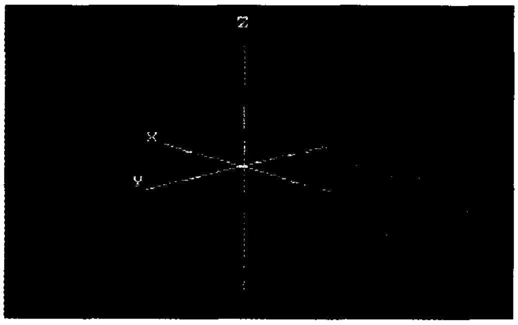
The first step in decoding is to acquire the serial bit stream. This can be done using a dedicated sample loop or, with a little more work, using interrupt processing. Since the idea is the same regardless of the details, I decided to use a sample loop in my demonstration program (Listing 1). The code simply records the incoming data stream and deposits the data in a sample buffer a byte at a time. Sampling begins when Card Present goes active and terminates when Card Present returns idle. Any incomplete byte that has been partially assembled at the time when sampling terminates is simply discarded. The abundance of leading and trailing zeros allows losing some bits at either end without causing any problems.

Once the sampling interval completes, control is transferred to the decoding algorithm. Presented in Listing 2, the track- 1 decode algorithm consists of nothing more than some initialization and the essence of the

CAN YOU SEE THIS PICTURE?



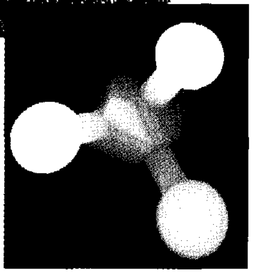
Why pay $30 to $50 for a 3-D Stereogram at your local Mall when you could *MAKE YOUR* ***OWN!***



***3-D STEREOGRAMS** and*
SOLID OBJECT MODELING

TARDIS Systems

901 18th Street, Suite 301
Los Alamos, NM 87544
(505) 662-9401 • fax (505) 662-6780
$39.95 plus S&h



System **Requirements: DOS PC, 386-486 with 4.0** MB RAM, **hard disk,** SVGA 640x480x256 color mode.

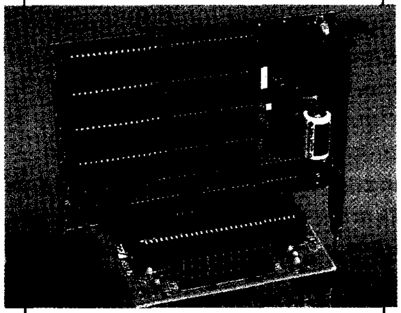
This full 3-D **object modeling code allows you to** create complicated 3-D scenes which can be viewed **from** any perspective or distance, and then **rendered into** your own **random-dotstereogram image.**

The system is mouse-driven, contains a basic sketch pad, a 3-D object editor and supports PCX and binary file output.

#136

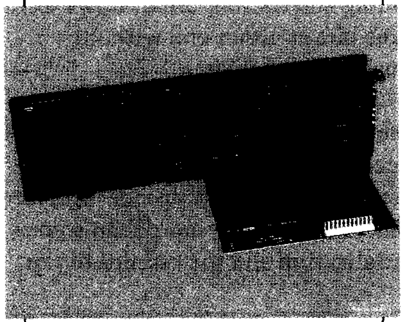
VMAX®

QUALITY PRODUCTS
RESPONSIVE SERVICE
RELIABLE DELIVERY



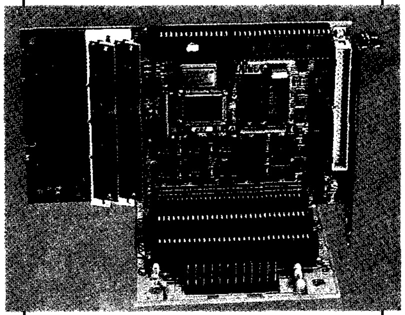
SOLID STATE DISK —$135*

½ **Card** 2 Disk Emulator
EPROM, FLASH and/or SRAM
Program/Erase FLASH On-Board
1M Total, Either Drive Bootable



25MHZ 386DX CPU — $695"

Compact AT/Bus or Stand Alone
On-Board SVGA, IDE, FDC, 2 Ser/Bi-Par
FLASH/SRAM Drives to 2.5M
Cache to 128K, DRAM to 48M



TURBO XT
w/FLASH DISK — $266"

To 2 FLASH Drives, 1M Total
DRAM to 2 M
Pgm/Erase FLASH On-Board
CMOS Surface Mount, 4.2" x 6.7"
2 Ser/1 Par, Watchdog Timer

All Tempustech VMAX products are PC Bus Compatible. Made in the U.S.A., 30 Day Money Back Guarantee *QTY 1, Qty breaks start at 5 pieces.

TEMPUSTECH, INC.

TEL: (800) 634-0701
FAX: (813) 643-4981

Fax for fast response!	295 Airport Road Naples, FL 33942

#137

Listing *P-continued*

```
                    MOV     R4,#0             ;initial LRC
                    MOV     R0,#0             ;bit synchronizer
                    CALL    FIND-START        ;main decode loop
                    JNC     L?M1D5            ;start bit error
                    CALL    GET-CHAR          ;Start Sentinel
                    JB      ACC.7,L?M1D5      ;format error
                    CJNE    A,#M1_SS,L?M1D5 ;start sentinel error
L?M1D2                                        ;data byte or End Sentinel
                    CALL    GET-CHAR
                    JB      ACC.7,L?M1D5      ;format error
                    CJNE    A,#M1_ES,L?M1D3  ;end sentinel not found
                    JMP     L?M1D4
L?M1D3
                    CALL    STORE-CHAR        ;data character
                    JMP     L?M1D2
L?M1D4                                        ;LRC
                    CALL    GET-CHAR          ;get LRC
                    JB      ACC.7,L?M1D5      ;format error
                    MOV     A,R4
                    JNZ     L?M1D5            ;LRC error
                    MOV     DPTR,#MD1_BUF     ;good return
                    MOV     A,R3              ;final character count
                    RET
L?M1D5              ;decode error, check if 1st pass
                    CJNE    R5,#1,L?M1D0      ;check direction
                    CLR     A                 ;bad return
                    RET
                    ENDPROC
```

Listing **3**—*The intermediate layer **of software** is one **level** removed from the nitty-gritty details.*

```
;Get the next bit from the sample buffer
;output: C contains data bit
GET-BIT             PROC
                    CJNE    R0,#0,L?GB1      ;bit synchronizer
                    MOV     R0,#8
                    PUSH    ACC
                    MOVX    A,@DPTR
                    CALL    INDEX_PTR
                    MOV     B,A
                    POP     ACC
                    DEC     R1                sample counter
L?GB1               XCH     A,B
                    CALL    POSITION_BIT
                    XCH     A,B
                    DEC     R0                bit synchronizer
                    RET
                    ENDPROC

;Find the first '1' bit in the samp e buffer
;output: C=1 if bit is found
FIND-START          PROC
L?FS1:
                    CJNE    R0,#0,L?FS2       bit synchronizer
                    MOV     R0,#8
                    MOVX    A,@DPTR
                    CALL    INDEX_PTR
                    DJNZ    R1,L?FS2          sample counter
                    JMP     L?FS4             out of samples
L?FS2:              CALL    LOCATE-BIT        test for a '1' bit
                    JC      L?FS3
                    CALL    POSITION_BIT
                    DEC     R0                bit synchronizer
                    JMP     L?FS1
```

(continued)

Listing *3-continued*

```
L?FS3:                                  ;good return
            MOV     B,A                 ;save copy in B
            SETB    C
            RET
L?FS4:      CLR     C                   :bad  return
            RET
            ENDPROC

;Get the next 7 bit element from the sample buffer
;output: ACC contains data element
,            error flag is ACC.7
GET_CHAR    PROC
            MOV     A,R1                ;sample counter
            JZ      L?GC2               ;out of samples
            MOV     R7,#7               ;bit counter
            CLR     A
L?GC1:      CALL    GET_BIT             ;next bit
            RRC     A
            DJNZ    R7,L?GC1
            RR      A
            JNB     P,L?GC2             ;parity error
            ANL     A,#3FH              ;discard parity
            PUSH    ACC
            XRL     A,R4                ;calculate LRC
            MOV     R4.A
            POP     ACC                 ;good return
            RET
L?GC2:      SETB    ACC.7               ;bad  return
            RET
            ENDPROC
```

(continued)

decode logic. Limiting the gyrations contained in the main body of this routine not only makes the logic easy to follow, but permits the same code to handle the decoding in either a forward or reverse direction.

The initial entry point assumes a forward decode attempt and sets up the necessary flags, pointers, and counters before jumping into the main initialization code. After initialization, the sample buffer is scanned for the first one bit, at which time a 7-bit element is assembled. If the parity is correct and the character code checks out to be a Start Sentinel, the code proceeds and starts pulling successive data elements from the sample buffer. If the data element is not an End Sentinel, the character is translated to ASCII and stored in the decode buffer. Should an End Sentinel be detected, the program extracts the next character, which is assumed to be the LRC byte, and finally checks the calculated LRC for a value of zero.

The checks and balances included in the execution of this loop include

Cross Assemblers

- Local Labelsand Cross Reference
- Powerful Macro Substitution Capability
- Machine Cycle Counting
- 32 Significant Character Labels and Symbols
- Unlimited include File Capability
- Selectable Intel Hex or Motorola Hex Object File

Simulators

- Source View Symbolic Debugging
- Attach Keyboard, Screen or Data Files to Simulate i/O
- Machine Cycle Counting
- Ten User-definable Screens
- Unlimited Breakpoints, Memory and i/O Mapping
- Trace File to Record Simulator Session

Disassemblers

- Automatic Substitution of Defined Label Names for Al Jumps and Branches
- Automatic insertion of Supplied Comments and Expressions

Application Source Libraries

- 16 and 32 bit integer Arithmetic and Numeric/String Conversion

PseudoCorp

716 Thimble Shoals Blvd.
Newport News, VA 23606

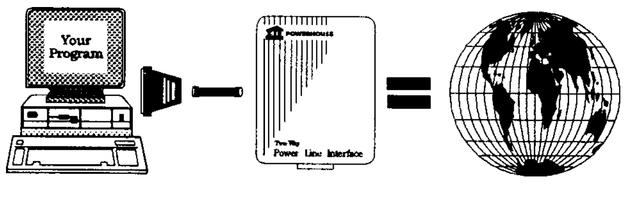
(804) 873-1947 FAX:(804) 873-2154
BBS: (804) 873-4838

#138

TW523 Power Line Interface

Developers Kit

Interface Your Computer To Transmit And Receive X-10 Codes Over Your AC Power Line.
Two-Way Communication.
Real Time Environment Control.



Kit Includes
TW523, Cable, Interface Connector (S/P)
Documentation. Source Code Supplied in "C", Pascal, BASIC or Run Time.
Disks 5.25in & 3.5in Format.

$65.00

a Baran-Harper Group Inc.

Voice (905) 294-6473 BBS (905) 471-6776
Fax (905) 47 I-3730

#139

things such as parity, a cumulative LRC, and checking to make sure I haven't run out of samples. If everything checks out, the program terminates and returns with DPTR pointing to the decoded data buffer and the character count contained in ACC. Should a decode failure occur, a test is performed on the direction flag and if this is an attempt at a forward decode, the routine jumps to the reverse initialization entry point. The reverse entry is similar to the forward decode entry but sets up the sample pointer to the end of the sample buffer and sets the direction flag to indicate a reverse operation.

The routines contained in the intermediate layer are shown in Listing 3. The meaning and operation of these routines should be apparent. The key routine in this section is GET_BIT, which picks off the next bit from the sample buffer, essentially restoring the sequential nature of the initial magnetic bit stream. FIND_START is used to synchronize with the first one bit. GET-CHAR first checks to make sure it hasn't run out of samples, then assembles the next 7-bit data element while doing a parity test and LRC calculation. Any problems encountered here are sent back to the caller and are handled there. STORE_CHAR translates and deposits the data character into its respective location in the decoded-data buffer and increments the character counter.

Listing 4 shows the low-level code. These routines perform the most rudimentary functions and operate in accordance with the direction flag.

Listing I-continued

```
;Translate and store the data character
;input: ACC contains data character
STORE-CHAR  PROC
            PUSH   DPL
            PUSH   DPH
            PUSH   ACC
            MOV    DPTR,#MD1_BUF
            MOV    A,R3            ;character counter
            ADD    A,DPL
            MOV    DPL,A
            CLR    A
            ADDC   A,DPH           ;generate displacement
            MOV    DPH,A
            POP    ACC
            ADD    A,#' '          ;translate
            MOVX   @DPTR,A         ;store
            POP    DPH
            POP    DPL
            INC    R3              ;character counter
            RET
            ENDPROC
```

LOW COST DEVELOPMENT TOOLS

We have a complete line of 'C' compilers (MICRO-C) for: 68HC08, 6809, 68HC11, 68HC16, 8051/52, 8080/85/Z80, 8086 and 8096 processors. Cross assemblers for these plus others. Source code and porting packages are available!

Development Kits: $99.95 + s&h
Includes C compiler, Cross assembler, ROM Debug monitor, Library source code, Editor, Telecomm program and everything else you need to do 'C' and Assembly language software development for your choice of processor.

Emily52, a high speed 8052 simulator: $49.95 + s&h
High speed (500.000 instructions/sec on 486/33). Hardware emulation mode accesses real ports/timers on your target system. Includes PC hosted "in circuit' debugger.

BD52, a complete 8052 Development System: **$249.95**+ s&h
Everything you need for 8052 development, including:
Hardware: 8032 based single board computer with 32K ROM, 32K RAM, RS-232 port, Hardware debug support.
Soflware: DDS MICRO-C Developers Kit, EMILY52 Simulator, PC Hosted "in circuit" debugger and kernal in ROM.

loaded **BD52:** $299.95 + s&h --
As above, with extra 32K RAM, 4 A/D + 1 D/A, 2K EEPROM, 7 line relay driver, watchdog and power monitor.,

Call or write for our free catalog.

DUNFIELD DEVELOPMENT SYSTEMS

MICRO-C

P. 0. Box 31044
Nepean, Ontario Canada
K2B 8S8
Tel: 613-256-5820 (l-5pm EST, Mon - Thu)
(BBS & Catalog Requests - 24 Hour with Touch Tone)
Fax: 613-256-582 1 (24 Hour)

Give the Gift of Technology...
12 issues for just $21.95*

Gift subscriptions to the Computer Applications Journal are available for the technologically inquisitive people on your holiday gift list.

It's hassle-free! Just give us a call and we'll set up your lucky recipient with a full one-year subscription to the Computer Applications Journal. They11 also receive a gift card with your name and the subscription's starting issue date.

To be sure your gift arrives in time for the holidays, call (203) 875-2199 or fax (203) 872.2204 before December 15,1993.

***Price good in U.S. only. Canada/Mexico $31.95, all other foreign $49.95. U.S. funds drawn on U.S. banks only**

INDEX_PTR either increments or decrements the sample pointer, POSITION_BIT likewise does either a right or a left shift, and LOCATE-BIT returns the state of the least- or most-significant bit of the accumulator.

GO AHEAD, TAKE A SWIPE

Let me touch on a few additional points that may not be immediately apparent before signing off. Storing the sampled data in a continuous stream makes the sample routine work equally well with the various bit configurations used for the different recording tracks. This would not be easily attained if you tried to generate a particular element format during sample time. Furthermore, if you look at the differences between the encoded character sets and the bit formats for the different tracks, you will find that they differ in only a few areas. With a few minor changes, such as the defined Start Sentinel, number of bits per element, and character translation method, the decode routine I've shown could easily be coerced to handle the decoding of any of the standard magnetic tracks.

As a matter of fact, by recoding and redefining the hard-coded constants as variables, these could be set up for the particular data track at execution time before invoking the decode function. Doing so would not only save program memory, but would also allow you to use a routine you were comfortable with. ◼

John Dybowski is an engineer involved in the design and manufacture of hardware and software for industrial data collection and communications equipment.

Listing 4—*The low-level routines get right down to the ground and take care of all the gory details.*

```
;Index the sample pointer either forward or backward
INDEX_PTR           PROC
        CJNE        R5,#0,L?IP1   ;check direction
        INC         DPTR          ;forward
        RET
L?IP1:  PUSH        ACC           ;backward
        DEC         DPL
        MOV         A,DPL
        CJNE        A,#-1,L?IP2
        DEC         DPH
L?IP2:  POP         ACC
        RET
        ENDPROC

; Position bit is in ACC into C in either a right or left shift
POSITION_BIT        PROC
        CJNE        R5,#0,L?PB1   ;check direction
        RRC         A             ;forward
        RET
 ?PB1:  RLC         A
        RET
        ENDPROC

  Locate a 1 bit, either msb or lsb; output: C=1 if bit is a one
LOCATE-BIT          PROC
        CJNE        R5,#0,L?LB1   ;check direction
        MOV         C,ACC.0       ;forward
        RET
L?LB1:  MOV         C,ACC.7       ;backward
        RET
        ENDPROC
```

CONTACT

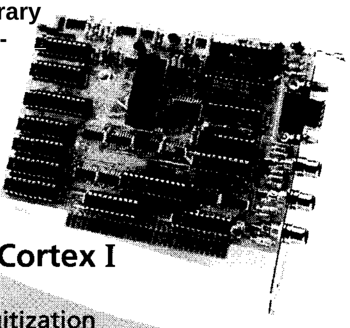
Mag-Tek, Inc.
20725 S. Annalee Ave.
Carson, CA 90746
(213) 631-8602
Fax: (213) 631-3956

IRS

419 Very Useful
420 Moderately Useful
421 Not Useful

Video Frame Grabber

- $495 Including Software with "C" Library
- Half Slot Card for Compact Applications-
- Real Time Imaging with Display Output
- 8 Bit (256 Gray Levels)



Cortex I

FEATURES:

- Single 512 X 484 Image or Four 256 X 242 Images
- External Trigger
- Input Look Up Table
- Low Power Option Available
- Elegant Software Interface
- <10 nsec Pixel Jitter Means Accurate Digitization
- EISA (PC) Bus and STD Bus Products Available
- RS-170 and CCIR Video Formats Available
- Binary and TIFF File Formats

The Cortex I is used in machine vision, industrial control, medical, security and scientific applications around the world.

ImageNation strives to delight customers with quality products and personal service at a competitive price.

Call today for volume pricing or to discuss your application.

ImageNation Corporation
Providing Imaging Solutions

P.O. Box 276
Beaverton, OR 97075
(503) 641-7408

FAX (503) 643-2458 • (800) 366-9131

#140

PATENT TALK

by Russ Reiss

In searching for patents relevant to this month's theme of "Programmable Devices," I came across two main categories. First up is a horde of devices that benefit by becoming programmable through the inclusion of microprocessors. The second group consists of memory devices such as EPROMs, EEPROMs, EPLDs, and the like which are the programmable components used in making these devices programmable. I have elected to report this month on nine patents chosen from both categories. The first five abstracts present some novel and interesting devices which benefit from their programmability; the last four abstracts present some recent developments in the creation of new types of memory devices.

Leading off, Abstract 1 presents an alternative to the often troublesome task of removing and upgrading an EPROM in a system when changes become necessary. Since this patent is assigned to Ford Motor Company, one can presume it is related to engine emission (or other) controls mounted (perhaps even within sealed packages) in the automobile. Certainly it is a time-consuming, and hence costly, matter to access and upgrade the program or parameters stored in these modules. The approach covered by this patent permits new code to be loaded instead into an EEPROM which is part of the system. An address comparator detects the section of old code that is to be redirected to the new code stored inside the EEPROM. Detection of this address causes program memory access to switch from the

1

Patent Number	4,905,200
Issue Date	1990 02 27
Inventor(s)	Pidsosny, Richard A.; Burke, Michael J.; Jarvis, Mark W.
Assignee	Ford Motor Company
US References	3,748,653 3,934,227 4,051,461 4,093,985 4,150,428 4,380,066 4,450,559 4,463,450 4,592,024 4,750,158 4,751) 703 4,757,475
Title	Apparatus and method for correcting microcomputer software errors
Abstract	An apparatus for replacing an undesired code stored in a plurality of storage locations in read-only memory (ROM) with a replacement code. The replacement code, including a jump-back instruction to readdress the ROM, are programmed into an EEPROM. First and second storage locations store the beginning address of the undesired code and the beginning address of the replacement code, respectively. When the beginning address of the undesired code is generated by a program counter, a comparator generates a transfer signal. In response, the beginning address of the replacement code stored in the second storage location is transferred into the program counter. Each address of the replacement code is then sequenced by the program counter until a central processing unit recognizes the jump-back instruction. The central processing unit then readdresses the ROM in response to the jump-back instruction after the entire replacement code has been executed in place of the undesired code.

2

Patent Number	4,986,145
Issue Date	1991 01 22
Inventor(s)	Mehta, Hemang S.
Assignee	Chrysler Corporation
US References	3,895,541 4,419,909 4,829,434 4,894,780
US Class	741843 741844 74/856
Int. Class	B60K 41 /04
Title	Method of engine model determination for use in an electronically controlled automatic transmission
Abstract	A method of engine determination is used in an electronically controlled automatic transmission system. A transmission controller is capable of determining the particular engine model by data sent by the engine controller. The method includes determining whether the data received equals a first predetermined engine model byte stored in RAM. If not, the method determines whether the data received is valid. If so, the method determines whether the data received equals a second predetermined engine model byte stored in EEPROM.

PATENT TALK

EPROM to the EEPROM. A "jump back" instruction returns execution to the original code after the "patch" is executed. Such a technique would seem to have applicability in many types of environments other than vehicles. Also, an ability to load the EEPROM remotely over a telecommunication link would further benefit this technique.

Abstract 2 from another automobile manufacturer, this time Chrysler Corporation, struck me as an interesting and novel use of programmability. It also points out how "electronic" the modern automobile is becoming. In this situation, the electronic control unit of an automatic transmission communicates with the electronic control for the engine to determine which type of engine it is connected to. Presumably, parameters such as shift points are altered depending upon which engine the transmission is paired with. Note once again the use of EEPROM to store variable data which (probably) can be updated whenever the need arises.

Yet again in the area of automobile control, Abstract 3 from Delco Electronics Corporation presents an antitheft device. Here again the ever-popular and dynamically changeable EEPROM comes to the rescue. In this instance, the value of a resistor pellet mounted in the ignition key is measured and compared with a value stored in the EEPROM. A match permits the car to start. Obviously, a bit more added intelligence would permit lockout control too. Together they would provide very inexpensive, yet effective, protection scheme. The use of EEPROM permits field changes in the key code so that keys may be changed whenever the car is sold, for example.

Getting away from automobiles, the next two patents relate to smart circuit breakers. While to most of us a circuit breaker is a pretty mundane device, the future holds programmable breakers with interesting intelligence built in. Abstract 4 from Westinghouse presents a microprocessor-based circuit breaker with a removable "rating plug"

3

Patent Number	4,990,906
Issue Date	1991 02 05
Inventor(s)	Kell, Curtis N.; Griffin, R.; Dikeman, John M.; Nemirovsky, Mario D.
Assignee	Delco Electronics Corporation
US References	4,148,372 4,438,426 4,672,225 4,713,660 4,755,815 4,791,280 4,804,856
US Class	340/825.310 340/825.320 340/825.340
Int. Class	H04B 1/00
Title	Programmable vehicle antitheft system
Abstract	A vehicle antitheft device is disclosed which includes an electronically erasable programmable read-only memory (EEPROM) which stores a modifiable code which must be matched by an input code in order to start the vehicle. The ignition key includes a resistor pellet, engaged by contact in the ignition lock assembly, which is measured to provide the input code. To avoid problems associated with intermittent contact engagement with the resistor pellet, circuitry is provided to control the resistor measurement cycle.

4

Patent Number	4,958,252
Issue Date	199009 18
Inventor(s)	Murphy, William J.
Assignee	Westinghouse Electric Corp.
US References	4,351 ,013 4,809,125 4,870,531
US Class	361/93 361 /87 364/483
Int. Class	H02H 3/08
Title	Circuit breaker with rating plug having memory
Abstract	A microprocessor-based circuit breaker has a removable rating plug which provides a reference for the current rating of the breaker. The removable rating plug also has an EEPROM to which the microcomputer of the circuit breaker writes for storage, the number of operations of the circuit breaker operating mechanism, and a trip history of the breaker in the form of a count of the number of trips weighted by the magnitude of the trip current. The rating plug is periodically removed and plugged into a portable programmer which selectively displays the number of operations and operating history of the circuit breaker. The programmer clears this operating data when recorded and can write into the EEPROM the style and serial number of the circuit breaker with which the rating plug is to be used.

which determines its trip point. Our friend the EEPROM once again comes into play. It is used in the removable plug to store such information as the number of operations of the breaker and at which levels of trip current they occurred. The plug may be removed and its stored information downloaded to a special unit which may also write information back to the plug in order to specify valid usage.

Abstract 5 from Square D Company presents another variation on the "smart circuit breaker" concept. In this case, the patent attempts to be considerably broad, discussing the use of the microprocessor and EEPROM memory to measure, calculate, store, communicate, and regulate many types of quantities and information.

As promised, the final three abstracts present some recent developments in the fabrication of programmable memory elements. While I am not personally aware of such devices being available currently, it will be interesting to see when they appear and how they will be used. Intel, in Abstract 6, presents a process for simultaneously fabricating both EEPROM and flash EPROM cells together on the same material. Such a dual-type memory element would present useful and interesting possibilities in compact system

5

Patent Number	4,996,646
Issue Date	1991 02 26
Inventor(s)	Farrington, Ronald L
Assignee	Square D Company
US References	4,419,619 4535,409 4,680,706 4,709,339 4,717,985 4,747,061 4,783,748 4,794,369 4,803,635
US Class	3641483 307/1 32E 3401657 361/71 3641492
Int. Class	G06F 15/56 GO1 R19/00
Title	Microprocessor-controlled circuit breaker and system
Abstract	A circuit breaker system uses a microprocessor for calculating at least one function of a measured current flow. The microprocessor provides other functions such as serial data stream communications, the ability of many circuit breaker systems to communicate with a central computer, storage of trip information concerning the last trip, storage of historical trip information concerning a number of past trips, EEPROM memory for storing trip information. The microprocessor may inhibit tripping on a high-current fault to permit storage of trip information into a memory. Power is derived from current transformers drawing energy from current flow to a load and the electronics are protected from high voltage caused by heavy current flow to the load. Optical isolators are used for the circuit breaker to communicate with external equipment, a multiturn resistor adjusts an external test voltage for testing the circuit breaker system, in the event that there is no load or a load drawing insufficient current to provide power for the electronics the circuit breaker may be externally powered to provide readout of electrically erasable programmable read only memory, the microprocessor reads setting switches and a multiplier plug. The microprocessor is capable of digitizing selected quantities frequently and of digitizing other quantities less frequently. A second microprocessor permits one microprocessor to sample voltage and current rapidly for metering purposes, and the second microprocessor operates other functions.

6

Patent Number	4,957,877
Issue Date	19900918
Inventor(s)	Tam, Simon M.; Lai, Stefan K.
State/Country	CA
Assignee	Intel Corporation
US References	4,203,158 4,517,732 4,527,259 4,701,776 4,745,083 4,782,424 4,804,637 4,822,750
US Class	437143 437144 437/52 4371193
Int. Class	HO1 L 21170
Title	Process for simultaneously fabricating EEPROM cell and flash EPROM cell
Abstract	Improved processing which permits the simultaneous fabrication of block erasable flash EPROM cells and individually erasable EEPROM cells. A polysilicon finger extends from the floating gate of the EEPROM cell over a tunnel oxide region. Doped regions are formed under this finger by implanting dopants in alignment with the finger during the implantation of the source and drain regions for the cells and then driving the dopant under the finger. The arsenic dopant used to form the source and drain regions for the cells is used to form the doped regions along with the phosphorus dopant used for the source region of the flash EPROM cells.

PATENT TALK

7

Patent Number	5,075,888
Issue Date	1991 1224
Inventor(s)	Yamauchi, Yoshimitsu; Tanaka, Kenichi; Sakiyama, Keizo
State/Country	JPX
Assignee	Sharp Kabushiki Kaisha
US References	4,615,020 4,672,580 4,721,987 4,760,556 4,786,954 4,813,018
US Class	3651228 3651149 3651185 357123.5
Int. Class	G11C 11/40
Title	Semiconductor memory device having a volatile memory device and a nonvolatile memory device
Abstract	A semiconductor memory device composed of a DRAM, an EEPROM, a mode switch means for selecting either mode of the DRAM mode and the EEPROM mode, and a transfer means for transferring data stored in the DRAM to the EEPROM and vice versa. The DRAM consists of one transistor and one capacitor, and one of the terminals of the capacitor is electrically isolated.

8

Patent Number	5,043,940
Issue Date	1991 08 27
Inventor(s)	Harari, Eliyahou
State/Country	CA
US References	4,087,795 4,181,980 4,279,024 4,357,685 4,448,400 4,652,897 4,667,217
US Class	65/1 68 365/1 85 3651218
Int. Class	G11C 11/56
Title	Flash EEPROM memory systems having multistate storage cells
Abstract	A memory system made up of electrically programmable read-only memory (EPROM) or flash electrically erasable and programmable read only memory (EEPROM) cells. An intelligent programming technique allows each memory cell to store more than the usual one bit of information. An intelligent erase algorithm prolongs the useful life of the memory cells. Use of these various features provides a memory having a very high storage density and a long life, making it particularly useful as a solid-state memory in place of magnetic disk storage devices in computer systems.

9

Patent Number	5,159,570
Issue Date	1992 1027
Inventor(s)	Mitchell, Allan T.; Tigelaar, Howard L.
Assignee	Texas Instruments Incorporated
US References	4,811,067 4,907,047
US Class	365/1 85 365/1 82
Int. Class	G11C 13/00
Title	Four memory state EEPROM
Abstract	An EEPROM memory cell having sidewall floating gates is disclosed. Sidewall floating gates are formed on sidewalls of a central block. Spaced apart bit lines are formed to serve as memory cell sources and drains. Sidewall floating gates are capable of being programmed independently of one another. When control gate is actuated and either bit line or bit line is used to read the device, four separate memory states may be identified depending on whether either, neither, or both of the sidewall floating gates have been programmed.

applications. In particular, I think they would make an intriguing memory card device which could be used to store both program and data concurrently. Also, such a device would directly support Ford Motor Company's code patching technique presented above in Abstract 1.

In a similar vein, Sharp presents in Abstract 7 a novel programmable memory device which is both DRAM and EEPROM. Presumably, the combination of the two devices into one unit offers the designer the benefits of both: fast access time and unlimited read/write capability of the DRAM coupled with low power and nonvolatility of the EEPROM. Keep a sharp eye for this one to hit the market.

Finally, the potential benefits of multivalue storage, in contrast to binary only, has long been known but seldom used. Abstracts 7 and 8 present two approaches to storing more than one bit of information per memory cell. Perfection and commercialization of such techniques could achieve a new horizon in storage capacity and memory system design. And as we approach physical speed barriers in component design, coupling such memories with multil evel processing elements could be one way to tackle the esign of yet faster computers. ❑

Russ Reiss holds a Ph.D. in EE/CS and has been active in electronics for over 25 years as industry consultant, designer, college professor, entrepeneur, and company president. Using microprocessors since their inception, he has incorporated them into scores of custom devices and new products. He may be reached on the Circuit Cellar BBS or on CompuServe as 70054,1663.

SOURCE

Patent abstracts appearing in this column are from the Automated Patent Searching (APS) database from:

MicroPatent
25 Science Park
New Haven, CT 065 11
(203) 786-5500 or (800) 648-6787

I R S

422 Very Useful
423 Moderately Useful
424 Not Useful

DO YOU NEED CONTROL ?

If you're looking for a temperature sensor that allows your computer to not only monitor the temperature but respond to it ,
look no further.
Temp-A-Chip,, is a solid state temperature sensor - providing truly linear measurement of temperature. The Temp-A-ChipTM is an intelligent, user configurable sensor which interfaces with your computer. No batteries are needed to operate the Temp-A-ChipTM , it plugs into any standard RS232 interface.

Temp-A-ChipTM

LCD Display	**Solid State Design**
No Batteries Req'd	**RS-232 Interface**
Easy To Install	**Easy To Use**

Can You Afford Not to Call Today?
(800) 274-8699
$149.00 $5.00 P & H

How Can You Catch Nasty Race Conditions That Take All Night to Happen Without Waiting All Night?

CodeProbe.TM

The new high-performance software analyzer that captures, time-stamps, and records software and hardware interrupts, DOS calls, BIOS interrupts, and user-defined events in real-time for analysis of race conditions, interrupt activity, and service times. CodeProbe gives you the hard facts you need to fix the big one that stands between system test and shipping your product.

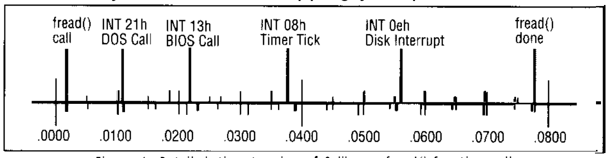


Figure 1. Detailed timestamping of C library *fread()* function call.

If **CodeProbe** can break-down a library function call into its components (above), imagine how you'll see context switches, device interrupts, and other asynchronous code. **Cdl *today for free technical specifications!***

GENERAL SOFTWARE TM
P.O. Box *2571,* Redmond, WA 98073
Tel 206.39 1.4285
Fax **206.557.0736**
BBS **206.557.4BBS**

#142

CONNECTIME

conducted by Ken Davidson

The Circuit Cellar BBS
300/1200/2400/9600/14.4k bps
24 hours/7 days a week
(203) 871-1988—Four incoming lines
Internet Email: @circellar.com

In case you missed my editorial on page 2, let me give you the exciting news here: Circuit Cellar is now accessible through Internet Email. Now before you get too excited, the BBS is not accessible via telnef or ftp. Right now the cost of providing such access is outrageous. Instead, you may send us mail through the Internet and also pick up magazine-related files using ftpmail.

Additionally, all users who call into the Circuit Cellar BBS via modem automatically get an Internet address and may send and receive Internet Email using the BBS as their gateway. Simply put a period between your first and last name, add our domain name to the end, and you have your Internet Email address. For example, my address is ken.davidson@circellar.com.

There is lots more you can do, too. For all the information, call the BBS and download the information file available on-line, or send an Email message to ftpmail@circellar.com and include the phrase "get he/p. txt" in the body of the message (not in the subject field). It will be mailed back to you automatically.

On to the messages at hand. We start out this month with a discussion about using 60-Hz transformers on 50-Hz lines and vice-versa. It's not an open-and-shut issue. We conclude with a look at how to control a small DC motor using a microcontroller.

Transformer Frequencies

Msg#:13133
From: RONALD HATCHER To: ALL USERS

Does anyone know what is involved in changing a 220-V, 50-Hz AC current to 120 V, 60 Hz?

Msg#:13718
From: LARRY G NELSON SR To: RONALD HATCHER

I have done it with a solid-state converter and also with a motor generator set. There is no way to change the frequency with a simple transformer. The application you have may not care about the frequency, in which case a transformer can be used.

Msg#:17027
From: PAUL PETERSEN To: RONALD HATCHER

Yes, you need a step-down transformer available at most any electrical wholesale house. Don't worry about the difference between 50 Hz and 60 Hz since the transformers are broad enough to accommodate both.

Msg#:17140
From: TOM MAIER To: PAUL PETERSEN

Yes, you do need to worry about 50/60 Hz when you select a transformer. Some are 50, some are 60, and SOME are broadband, designed to handle both. If you connect a 60 to a 50-Hz line, you will end up with smoke.

Msg#:17524
From: PELLERVO KASKINEN To: TOM MAIER

I want to qualify your smoke statement. The frequency rating alone is not enough, although your choice is basically correct: the 60-Hz transformer is endangered on a 50-Hz line. But that is, if they have identical voltage ratings and then you apply exactly the same FULL voltage. The rule is that the magnetic path can take a certain amount of magnetic flux, depending on the size of the core. Given this number (which, by the way, is dependent on the cross-section and the length of the path due to leakage effects and, more than anything, it is dependent on the material of the core), you can apply up to a maximum magnetizing CURRENT to a winding of any given number of turns.

All this is in DC sense. When we start talking about transformers, we are also talking about AC and we tend to prefer talking about voltages instead of currents. Now, the rules get changed a little, just like a special kind of transformer [pulse transformer) is actually catalogued by the volt-second figure it can handle. Either a higher voltage or a longer time will bring it to saturation.

The inverse of time is frequency, so we can see that the possible voltage rating for any given transformer is directly proportional to the frequency (if we want to bring it to the edge of saturation). Therefore, a given 50-Hz transformer will always sustain the same amount of voltage at 60 Hz or it can even be rated 20% higher. But the opposite direction of conversion of course requires derating.

All of this is assuming a razor sharp saturation and a design driven to the leading edge in materials and weight saving. But most magnetic materials in use have quite rounded saturation characteristics and most transformers have had to be designed with a voltage range allowance, say ±10%. With this and a checking of the available voltage, I might sometimes conclude that it is safe for *me* to connect a 60-Hz-rated transformer to the 50-Hz line in Europe. And actually, there are both core losses and copper losses to consider. If the transformer secondary load is negligible, maybe it would tolerate some amount of saturation-caused increase in the primary current as well as the higher core temperatures.

Going the other way round-using a transformer at elevated frequencies-the main issue is the eddy current losses at the higher frequencies increasing according to a square law, *if we increase the voltage proportional to the frequency.* However, if we keep the voltage the same, the eddy current losses do not grow that bad. I would say most any ordinary 50-Hz power transformer can run quite well at 400 Hz. Maybe some transformers that have their cores welded for cheap manufacturing start suffering a little more than those with hand-stacked laminations and insulated screw assembly, but the difference is still minimal. However, I would not try to use an ordinary power transformer even at reduced voltages as an output transformer for an audio amplifier.

Here is again the formula that governs transformer designs (after you have picked a core):

$$n = V / (4.44\, f\, b\, A)$$

where

- n is the number of turns in the winding (each individually)
- V is the RMS voltage of sinusoidal excitation
- f is the applied frequency in Hz
- b is the *peak* magnetic flux density allowable in teslas (10 kG)
- A is the cross-section of the core in square meters

The 4.44 happens to be the product of pi and square root of 2, for those curious about this detail.

Another detail, that would need to be covered for a real treatise of the transformer design is the selection of the core based on the power requirement. Mostly it is handled by picking the value from the data sheet of a magnetic material manufacturer. But there is a way to estimate it, if you are willing to make fully custom coil former as well. The rule is that the cross-section of the iron core should be about as much as the "window" space for all the windings combined. This leads into iteration: Assume some cross-section, see how many turns it would require, and see if that many turns fit into HALF the window size (assuming you start the calculation with the primary that takes half of all winding space, with all the secondaries sharing the other half). Assume each wire turn occupies a space that is the square of the insulated wire diameter and that the wire can handle a conservative 1.5 A/mm^2 or a little more demanding 2.5 A/mm^2. Adjust your guess for the core cross-section and window size, and repeat until you reach an optimum design.

Sorry to become so wordy, but I think this issue is of such universal importance to any electronic designer to understand that I just could not stay away.

Msg#:17528
From: TOM MAIER To: PELLERVO KASKINEN

I was an engineer for four years at a company that made machines for world wide market. I know too well what happens if the wrong transformer gets used.. . the engineer gets his butt kicked!

Msg#:17795
From: PELLERVO KASKINEN To: TOM MAIER

By now I have been in a similar position for 15 years. So far no kicking to the butt, but then I think this issue has always been dear to me. On the other hand, the purpose of my previous message was the benefit of the general audience as I said.

Actually, I have started suspecting that the flow of information on this board is so rapid that many people cannot find the topic after a few months even with the subject search. I keep getting so much from the discussions here that I feel a need to contribute and to stay with my style, go down to the details where I can. So, in this case, for instance, my recollection was that this transformer topic was covered too long a time ago to have stayed accessible. That's why you got the message that you probably did not need. And that people who just call to ask a question and then call back to read the answers directly addressed to them are still going to miss it-too bad!

Msg#:18472
From: TOM MAIER To: PELLERVO KASKINEN

You gave a great answer. At one company I worked at we shipped around the world and occasionally we would assign the wrong transformers. I got caught in that once and it is now burned into my memory that 50 Hz is very different from 60 Hz.

Msg#:17799
From: DAVID MEED To: TOM MAIER

I think you just have to derate the transformer 50/60 of its power rating..

Msg#:18746
From: TOM MAIER To: DAVID MEED

Wrong.. . With no load you can burn up a 60-Hz transformer by connecting it to 50-Hz power. I've seen it done with 20-kVA transformers. Sizzle, sizzle, smoke, POW!

The impedance of the primary is not right at different frequencies. With a 60-Hz coil run at 50 Hz, you get large current flow in the primary, higher than it's rated for. This is why the transformer gets hot and fries it primary windings and shorts out. You can, of course, design the transformer to handle this, but it increases the cost. Therefore, if you are designing a cost-sensitive application for U.S. or European release only, then it is more economical to use the transformer for your frequency of interest. Shipping weight is also important on quantity items.

The international company that I worked for finally decided it wasn't worth the headaches and went with the heavier, more expensive 50/60 primary isolation transformer. The increased cost of the transformer was cheaper than the cost of stocking two of them and trying to decide with products got which transformers. Two different stock items and two different resulting products resulted in a paperwork increase and occasional shortage of one product and too much of the other.

If it doesn't said 50/60 on the label, then don't trust it to be both.

Also, you have to watch your motors and relays and solenoids. Same story on those. If it doesn't specifically say 50/60 and you plug a 60 into a 50 power, you get smoke. Plug a 50 into 60 and you get reduced power out (solenoids won't activate, motors have half torque, transformers have reduced output).

DC Motor Control

Msg#:16224
From: MARK **RUBINO** To: ALL USERS

I would appreciate being pointed in the right direction. I would like to learn about the basics on motor control (small DC motor) with a microcontroller. I figured it may be something like this: DAC to preamp (maybe another preamp) to final drive amp. Nothing fancy to start with. Books or literature that are light in math, and more for the technician. Thanks for the help.

Msg#:16621
From: PELLERVO **KASKINEN** To: MARK RUBINO

I can only cover some basics from the motor end, not so much the computer connections or algorithms.

Your assumption about a DAC to preamp and power stage is basically correct, even though the DAC can be implemented in direct pulse-width-plus-filtering fashion instead of the R-2R ladder fashion in some cases. In other words, the processor is using a timer instead of a parallel port to drive the speed command line. You can get the basics from a Hewlett-Packard motor control chip data sheet or from a similar publication by Galil. Also, National Semiconductor has something more remotely related to the topic. Sorry, I did not remember these chips earlier so I could have picked the part numbers for you.

As to the small DC motors, we would need to define at first the salient points of your interest. Exactly how small? What speed range? What speed accuracy? All of these affect the preferred solution very strongly. Also, are you talking of a pure speed control or also a position control? And does the position have to be absolute within one session, absolute even after power down, or is it just relative "now that we are here, move so much to this direction." All of the position-related stuff needs some sort of encoder or other position feedback sensor. For a computerized system, encoders are a natural choice, while for a direct amplifier solution, a potentiometer would do just fine. And for the encoders, a normal solution is to have a low-cost incremental encoder plus possibly some limit switches for establishing a home position [for one session only; must be repeated after each power up). Some fancy systems use either an absolute encoder or a resolver to establish a known position even through power down.

Then to the motor. I am not going to touch the brushless variety, just the plain old brush-type motor with a permanent magnet field. The simple mathematical basis for them is that they *generate* voltage when they run proportional to the strength of the magnetic field and the rotation speed as well as the number of wire turns in the armature winding. This generated voltage is called back EMF. Now, there is also some resistance in the armature winding and a small drop in the brushes/commutator. This drop is more or less constant, just like that on semiconductor diodes, though generally smaller, about 0.1 to 0.2 V. And then, something supplies the driving power, say we have a battery for the beginning.

One more principle for the preparations: a motor develops torque proportional to the . current * (and to the magnetic field). So what happens when you connect this small motor to a mechanical load and to the battery is this: At first, the motor speed is zero, no counter EMF. The current drawn from the battery is (battery voltage -brush drop)/(circuit resistance).

The circuit resistance may consist of additional elements besides the armature, but we get back to the importance of the armature resistance shortly. With this current, the motor develops torque and starts accelerating. I don't complicate the presentation with any changing torque requirements such as fans or blowers. But the principle is that the acceleration continues until the motor torque just matches the load torque requirement.

All along, the motor develops higher and higher counter EMF. With that, the *apparent* battery voltage drops, so the current drops. Let's repeat the formula: Current = (Battery voltage – Brush Drop – EMF)/(Circuit

Resistance). And the other formula: Output Torque = (Constant K times Current). The constant K is winding and magnet strength dependent, but constant for each motor.

OK, back to the speed CONTROL issue. You need to know the actual speed in order to control it intelligently. You have several options. A tacho generator is one. The pulse frequency from an encoder is another possibility. And typically the resolver interfaces produce a speed output signal as well. But if you do want to make it really small and simple, you do not have the luxury of separate speed-measuring devices.

Remember the counter EMF. It is proportional to the speed. But the terminal voltage to the motor is not the same as the EMF. No, it is the sum of the EMF and the product of current and armature resistance. So, you just have to know the armature resistance and the actual current to be able to calculate the EMF from a measured terminal voltage. You get the current by inserting a small resistance in series with the motor and measuring two voltages instead of the terminal voltage alone. Typically this is done in the analog domain, but if you have a computer with an A/D converter (in addition to the DAC), then you can do it digitally.

I hope this covers some of the issues, even though I did not give any sources for reading. All this comes just from my own head and is based on something I have had to learn in the past over and over again, so I could possibly quote them while sleeping. :-}

Msg#:16668
From: ED **NISLEY** To: PELLERVO **KASKINEN**

As an aside on absolute position sensing, the June "HP Journal" showed a truly neat trick for sensing the position of the follower on a leadscrew. The thing you want to avoid is running the follower all the way to the end every time you power up.. .but you don't want to spend big bucks for an absolute encoder with lots of precision.

The encoder uses three gears: one driven by the motor and two idler gears. Each gear has a hole sensed by a simple optical interrupter. The trick is the number of teeth on each gear: they're relatively prime so the same alignment won't come up twice in the length of the leadscrew. The example they used was a driver gear with 23 teeth and idlers with 24 and 25 teeth. The positions repeat every 600 (= 24 x 25) revolutions of the driver.

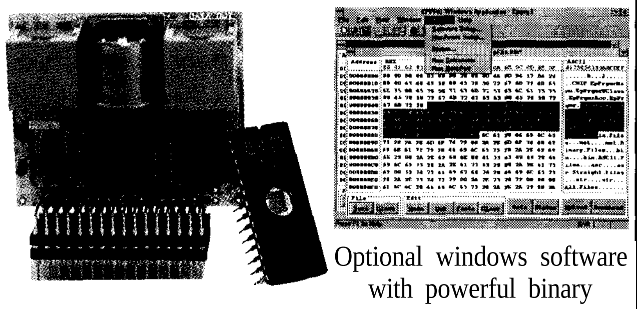
WORKS

Does your Big-Company marketing department come up with more ideas than the engineering department can cope with? Are you a small company that can't afford a full-time engineering staff for once-in-a-while designs?

Steve Ciarcia and the Ciarcia Design Works staff may have the solution. We have a team of accomplished programmers and engineers ready to design products or solve tricky engineering problems. Whether you need an on-line solution for a unique problem, a product for a startup venture, or just experienced consulting, the Ciarcia Design Works is ready to work with you. Just fax me your problem and we'll be in touch.

Remember...
a Ciarcia design works!
Fax (203) 871-8986

PROMJet Ultra Compact EPROM Emulator



Optional windows software with powerful binary editor Available ($50US).

- Emulates 64 Kbit to 8 Mbit 85/40ns EPROMs.
- Accepts Binary, Ext. Intel & Motorola formats.
- Fast download from printer port (1 Mbit/Sec).
- Fits into EPROM socket to minimize noises.
- Double sided SMT (size: 2.2"x0.7"x1.9").
- Jumperless configuration through software.
- Ni-Cd battery backup. Power-up emulation.
- Cascadable to 128 bits. Generates RESET+/-.
- Comes complete with software and cables.

15 DAY MONEY-BACK GUARANTEE.

PROMJet 64K-1M/2M	**85ns**	**$195/295**
PROMJet 64K-4M/8M	85ns	$4951695

WESTEC *Research Corp.* (213) 664-8909
2750 Riverside Dr. #205, LA, CA 90039

#143

CONNECTIME

To sense the absolute position (on power-up, for example) you turn the driver gear one revolution. You know when each hole shows up and thus the position of each gear. That gives you the "turn number" because those positions are unique within the 600-turn sequence. Bingo.. .you now know where the follower is positioned on the leadscrew! Because you make only a single leadscrew rotation, the follower doesn't move very much and you don't have to worry about smashing into anything.

(I think you might have to make slightly more than one turn to guarantee that each hole passes under its sensor.. .or perhaps the lack of a hole tells you where it must be!)

From the way they describe it, it's surely patented and all that, but it's a truly neat hack nonetheless.

Msg#:18176
From: LARRY G NELSON **SR** To: PELLERVO KASKINEN

One technique I have used in the past is to drive the DC motor with a PWM signal to control the speed. With the PWM output off, you can measure the voltage across the motor and read the back EMF. This voltage is proportional to the motor speed and can be used for control.

The *only* 8051/52 BASIC compiler that is

100 % BASIC 52 Compatible

and

has full floating point, integer, byte & bit variables.

- *Memory mapped variables*
- *In-line assembly language option*
- *Compile time switch to select 805 1/803 1 or 8052/8032 CPUs*
- *Compatible with any RAM or ROM memory mapping*
- *Runs up to 50 times faster than the MCS BASIC-52 interpreter.*
- *Includes Binary Technology's SXA51 cross-assembler & hex file manip. util.*
- *Extensive documentation*
- *Tutorial included*
- *Runs on IBM-PC/XT or compatibile*
- *Compatible with all 8051 variants*
- ***BXC51 $ 295.***

508-369-9556
FAX 508-369-9549

Binary Technology, Inc.
P.O. Box 541 • Carlisle, MA 01741

#144

Msg#:18296
From: PELLERVO KASKINEN To: LARRY G NELSON SR

I have used the back EMF as well for feedback. It is applicable, when the inertia is high enough and the inductance in series with the motor armature is low enough. Actually, one more condition is required. It is that the driver side can be put into a high-impedance state. Then, when you wait until the inductive current has decayed, you get the speed proportional and properly dependable signal. Good for a couple of percent linearity and accuracy.

The complications for my designs seem to be related to the efficiency of the motor when the current is periodically interrupted, jerks in speed when the inertia is small, and when the standard series inductance is pretty large.

Msg#:18260
From: JOHN **STIRTON** To: MARK **RUBINO**

An easy explanation of how to control motors is given in Chapter 13, "Robot Locomotion with DC Motors," in the "Robot Builder's Bonanza" (book) by Gordon McComb. I have built the circuits on pages 101 and 103, and they work very well. In his design, Mr. McComb uses power MOSFETs, which are adequate, however, a number of manufacturers make single IC H-bridge controllers. Sprague and RCA (SK brand) are just two manufacturers.

We invite you call the Circuit Cellar BBS *and exchange messages and files with other Circuit Cellar readers. It is available 24 hours a day and may be reached at (203) 871-1988. Set your modem for 8 data bits, 1 stop bit, no parity, and 300, 1200, 2400, 9600, or 14.4k bps. For information on obtaining article software through the Internet, send Email to ftpmail@circellar.com with the phrase "get help.txt" in the body of the message (not in the subject field).*

ARTICLE SOFTWARE

Software for the articles in this and past issues of *The Computer Applications Journal* may be downloaded from the Circuit Cellar BBS free of charge. For those unable to download files, the software is also available on one 360K IBM PC-format disk for only $12.

To order Software on Disk, send check or money order to: The Computer Applications Journal, Software On Disk, P.O. Box 772, Vernon, CT 06066, or use your VISA or Mastercard and call (203) 8752199. Be sure to specify the issue number of each disk you order. Please add $3 for shipping outside the U.S.

425 Very Useful 426 Moderately Useful 427 Not Useful

STEVE'S OWN INK

Lest We Forget that Chips Can...and Will

Everyone who knows me knows I have a favorite programming language. But it's not easy, or economical, to program in just one language anymore. It used to be that you could build the keenest computing device with just one VW bus full of parts, about 15 pounds of solder, 10 spools of different colored wires, and any 700-watt supply...But the times, they are a changing.

Now I can cram the equivalent of that whole VW full of parts into a couple of square inches. And miles of connections can be replaced with a couple dozen two-inch traces between megasmart chips. The trouble is now I have to program *everything.* When the first generation of programmable silicon appeared, you still had to use dozens of discrete logic chips and "program" the system with yards of colored wire. And if the system didn't work right, I could at least verify the connections between the components. Usually, after correcting the wiring errors, everything would be fine. One of the best features of discrete logic design is that testing it is pretty straightforward. An AND gate always acts like an AND gate. Nowadays, however, a chip can be programmed to act like anything!

I watched while one of my friends built his newest brainchild. This chap was busy programming nearly every component that was part of his project. He was programming the processor in C and assembly language. He was also programming several of his logic components in a variety of HDLs. Certain parts download their behaviors on power up, and others' *behaviors* were held in an EPROM image inside the parts. The design of this system was inside of the parts *and* in the way they were connected. He had a host of programming environments, a bench full of programmers, and a variety of conversion instruments so that all of these parts could be given its dose of intellect. How does he test a design where every chip is programmed? With other programs (simulators), or with programmable test jigs. This works great as long as the only bugs he has to deal with are his own.

I remarked at how his project seemed to be missing the characteristic snarl of wire that I associate with prototypes. He looked at the pristine landscape of his creation and grinned. "Steve, don't let appearances fool you. It might look like this on the surface. But that is only because the intractable mass of wires and cables have been replaced with a few thousand lines of code."

I guess things really aren't that different after all. When my old projects began to outgrow the circuit board, I would add another card for the new parts and patch the new circuits to the old ones with a few yards of colored wires. As his projects grow, he does the same thing. Only he just adds gates in increments of a-sea-of-gates at a time, and all his patches are done in code. And I suppose it's just as easy to find a missing or misdirected line of code as it is to find a missing or wrong connection in the wiring.

But, the biggest problem with programmable hardware is environmentally generated software. That's what happens when your device gets too close to the ventilation motor and the RFI reprograms your parts for you. When this happens, your chips can completely forget what you told them, or worse yet, they can learn something completely new and exciting! In either case, now your code is running along with a new subroutine called chaos. This kind of situation is pretty hard to fit into the design-for-test paradigm, even if you could model it in the first place. Is there any test method in existence that can trigger on and correct the behavior of misdirected electrons? Sure, the chances of quantum programming are slim, but there is an unfortunate corollary to Murphy's law that states the first occurrence of a quantum-well electron failure will happen at a million dollar account or in a satellite orbiting Mars.

Steve