

EMBEDDED PC MONTHLY SECTION

CIRCUIT CELLAR[®]

THE COMPUTER APPLICATIONS JOURNAL

#85 AUGUST 1997

EMBEDDED PROGRAMMING

Genetic Algorithms for FPGAs

Breaking the
64-KB Barrier

Flash Memory
Goes Serial

EPC: Video and PC/104

P U T S

N

T

I

S

W

H

I

L

E

I

F

L

T

S

C

E

C

H

A

R

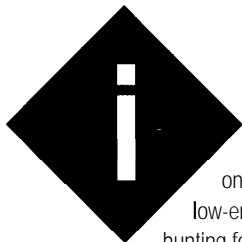
KEY



\$3.95 U.S.
\$4.95 Canada

TASK MANAGER

Let Me Tell You...



just finished an interesting conversation with one of our readers. He's an engineer with some low-end embedded experience who has found himself hunting for a job. To make himself more marketable, he wants to become familiar with some of the more common development tools and target processors on the market. What direction would I suggest he take?

An admirable goal, but talk about an open-ended question! Another common-and similar-question I get is, "I want to get into embedded control. Can you suggest a book to get me started?"

How do I respond to these types of questions? I've been away from the beginner end of this market far too long to offer first-hand suggestions. And asking about the most useful processor or tool to learn is a bit like asking, "Should I buy a minivan or a station wagon?" It depends. What are you wanting to do?

The ultimate answer I end up giving to a lot of people is, "Log on to the Circuit Cellar BBS and ask other users." In the past, I've promoted the BBS as a valuable source of technical knowledge, and I continue to do so. You, the readers out in the trenches, are the best source of advice I can suggest to other engineers.

Another great source is, of course, the articles found in these very pages. We start off this Embedded Programming issue with a discussion by Ingo Cyliax on genetic algorithms and how they can be implemented on an FPGA. Nature's been at it for millions of years, so there must be something to it.

Next, Jim Sibigroth revisits a design technique long used in the 8-bit world: bank switching to access more than 64 KB of memory. He gives some good advice for both hardware and software developers.

Speaking of software development, Brian Millier presents a powerful and easy-to-use development hardware/software combination for the Motorola 68HC705. Perhaps the tools will finally catch up to the targets in power and sophistication.

Finally, Gordon Dick presents an intelligent motion controller for DC motors.

Moving on to Embedded PC, Ralph Birt and Khoi Hoang survey the flat-panel display market. Edward Steinfeld examines the now ubiquitous Web browser as an affordable, powerful, and standardized graphical front end to embedded designs. On the PC/104 front, Richard Hopkins tells us how to force video data through the PC/IO4 bottleneck. Finally, Fred hauls in some water-meter data via an Internet appliance.

In our columns, Jan Axelson finishes up her MicroSeries on serial memory, Jeff adds some ultrasonic transducers to his robot, and Tom checks out some high-power serial flash memory.

editor@circuitcellar.com

CIRCUIT CELLAR[®]

THE COMPUTER APPLICATIONS JOURNAL

EDITORIAL DIRECTOR/PUBLISHER
Steve Ciarcia

ASSOCIATE PUBLISHER
Sue Hodge

EDITOR-IN-CHIEF
Ken Davidson

CIRCULATION MANAGER
Rose Mansella

MANAGING EDITOR
Janice Hughes

CIRCULATION CONSULTANT
John S. Treworgy

TECHNICAL EDITOR
Elizabeth Laurençot

BUSINESS MANAGER
Jeannette Walters

ENGINEERING STAFF
Jeff Bachiochi

ADVERTISING COORDINATOR
Valerie Luster

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
Rick Lehrbaum
Fred Eady

NEW PRODUCTS EDITOR
Hal-v Weiner

ART DIRECTOR
KC Zienka

PRODUCTION STAFF
John Gorsky
James Soussounis

CIRCUIT CELLAR INK[®], THE COMPUTER APPLICATIONS JOURNAL (ISSN 0696.6965) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (860) 875-2751. Periodical rates paid at Vernon, CT and additional offices. One-year (12 issues) subscription rate U.S.A. and possessions \$21.95. Canada/Mexico \$31.95. all other countries \$49.95. All subscription orders payable in U.S. funds only, via international postal money order or check drawn on U.S. bank.

VISIT OUR WEB SITE FOR SUBSCRIPTION INFORMATION AT www.circuitcellar.com

Direct subscription orders and subscription related questions to Circuit Cellar INK Subscriptions, P.O. Box 698, Holmes, PA 19043-9613 or call (800) 269-6301. POSTMASTER, Please send address changes to Circuit Cellar INK, Circulation Dept., P.O. Box 698, Holmes, PA 19043.9613.

Cover photograph Ron Meadows
PRINTED IN THE UNITED STATES

For information on authorized reprints of articles, contact Jeannette Walters (860) 875-2199.

HAJAR ASSOCIATES NATIONAL ADVERTISING REPRESENTATIVES

NORTHEAST & MID-ATLANTIC
Barbara (Best) Curley
(561) 694-2044
Fax: (561) 694-2051
B.Best-Hajar@worldnet.att.net

MIDWEST&SOUTHEAST
Christa Collins
(954) 966-3939
Fax: (954) 985-8457
HajarChrista@worldnet.att.net

WEST COAST
Barbara Jones
& Shelley Rainey
(714) 540-3554
Fax: (714) 540-7103
shelley.hajar@worldnet.att.net

Circuit Cellar BBS-24 Hrs., 2400/9600/14.4k bps, 6 bits, no parity 1 stop bit, (660) 871-1988. For information, mail to info@circuitcellar.com. World Wide Web: www.circuitcellar.com

All programs and schematics in Circuit Cellar INK[®] have been carefully reviewed to ensure their performance is in accordance with the specifications described, and programs are posted on the Circuit Cellar BBS for electronic transfer by subscribers.

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

Entire contents copyright © 1997 by Circuit Cellar Incorporated. All rights reserved. Circuit Cellar INK is a registered trademark of Circuit Cellar Inc. Reproduction of this publication in whole or in part without written consent from Circuit Cellar Inc. is prohibited.

- 12 Genetic Algorithms for FPGAs
Ingo Cyliax
- 18 When 64 KB Isn't Enough
Jim Sibigtroth
- 24 Windows-based Development System for the 68HC705C8
Brian Millier
- 62 Test Drive a Precision Motion Controller
Gordon Dick
- 70 **MicroSeries**
Using Serial EEPROMs
Part 2: Putting It All Together
Jan Axelson
- 76 From the Bench
It Can't Be A Robot
Part 3: It's Blind as a Bat
Jeff Bachiochi
- 82 Silicon Update
Serial Flash Busts Bit Barrier
Tom Cantrell

Task Manager
Ken Davidson
Let Me Tell You...

Reader I/O

New Product News:
edited by Harv. Weiner

Advertiser's Index 6

Priority Interrupt 9
Steve Ciarcia
The Fast Track

INSIDE ISSUE 95

EMBEDDED PC

- 36 Nouveau PC
edited by Harv Weiner
- 40 Flat Panels for Embedded PCs
Ralph Birt & Khoi Hoang
- 45 Web GUIs for Embedded Applications
Edward Steinfeld
- 49 PCQ PC/104 Quarter
Video on the PC/104 Bus
Richard Hopkins
- 55 APC Applied PCs
Internet Appliance Development
Part 2: Getting Flow-Meter Data
Fred Eady

www.circuitcellar.com

READER I/O

IMPROVING THE TIMING EDGE

I enjoyed Daniel Patten and Michael Miller's article ("A Universal IR Remote-Control Receiver," *INK* 82).

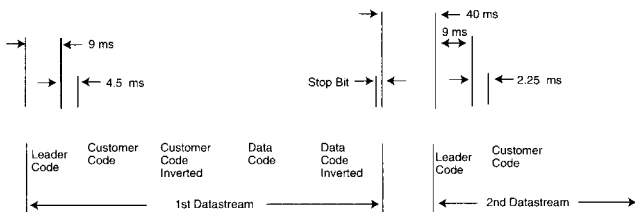
One thing I want to point out, though, is that the NEC-style data format shown in Figure 1 was not complete. Each time it is activated, the NEC transmitter/keyboard encoder chip sends out at least two datastreams with 40-ms interval in between. The customer and data codes of the two streams are the same. The difference is in the leader code.

The first stream's leader code consists of 9 ms high followed by 4.5 ms low. The second stream's leader code consists of 9 ms high followed by 2.25 ms low. The data format is shown below.

As you see, there is a 0.6-ms high after the 32-bit datastream. Although NEC didn't label this bit, it could serve as a stop bit.

I like *Circuit Cellar INK* a lot. Keep up the good work!

Joe Zhu
Johnston, RI



NARROWING SEARCH PARAMETERS, CAPTAIN!

First, let me say that I love *INK*. I've been a subscriber since June '92, and I find the magazine quite enjoyable. I've learned a great deal from reading it and even convinced several coworkers to subscribe as well!

I understand that creating an online comprehensive index that enables readers to search by author, subject, title, and so on takes a great deal of time and effort. However, this index would be extremely valuable to your subscribers, and I eagerly await its arrival.

I keep all my old issues of *INK*, but locating a specific article is extremely difficult given some of the creative titles, which scarcely indicate the article's content.

Until the comprehensive index can be created, why not give us access to a text version of the *INK* index so we can search the titles using the search feature on our word processors? Finding what we need in the 21-page two-column PDF file is time consuming and frustrating.

Brad Claflin
Austin, TX

Flipping through 21 pages of hardcopy is a hassle. Until we have an online search system up and running, search the index PDF file on your computer via the find feature in Adobe's Acrobat Reader. It's not so different from the find feature on your favorite word processor.

INK's index is available in PDF format on our Web site at <www.circuitcellar.com/public/cc/backissues.html>. There's also a link to the Adobe site for downloading the Acrobat tools you need to read the file.

Editor

DID EMILY POST INVENT E-MAIL?

I agree with everything Ken said in "Life's Little Mysteries" (*INK* 83) about voice phoning and more! I am incensed when I'm cut off in a face-to-face transaction so whoever I'm talking to can answer the phone. It's even worse when they spend my time conducting business that's obviously more important than mine.

E-mail has become my most efficient and pleasant communications medium. No more telephone tag. I take my turn along with anyone else whose message is queued ahead of mine. And, I can look over a reply in its proper order within my scheduled and unscheduled priorities!

Walt Boyd
wboyd@netdex.com

Contacting Circuit Cellar

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

Mail: Letters to the Editor may be sent to: Editor, Circuit Cellar INK, 4 Park St., Vernon, CT 06066.

Phone: Direct all subscription inquiries to (800) 269-6301.

Contact our editorial offices at (860) 875-2199.

Fax: All faxes may be sent to (860) 871-0411.

BBS: Editors and regular authors are available to answer questions on the Circuit Cellar BBS. Call (860) 871-1988 with your modem or telnet to bbs.circuitcellar.com.

Internet: Letters to the editor may be sent to editor@circuitcellar.com. Send new subscription orders, renewals, and address changes to subscribe@circuitcellar.com. Include your complete mailing and E-mail addresses in all correspondence. Author E-mail addresses (when available) may be found at the end of each article.

WWW: Point your browser to www.circuitcellar.com.

FTP: Access ftp.circuitcellar.com for article files.

NEW PRODUCT NEWS

Edited by Harv Weiner

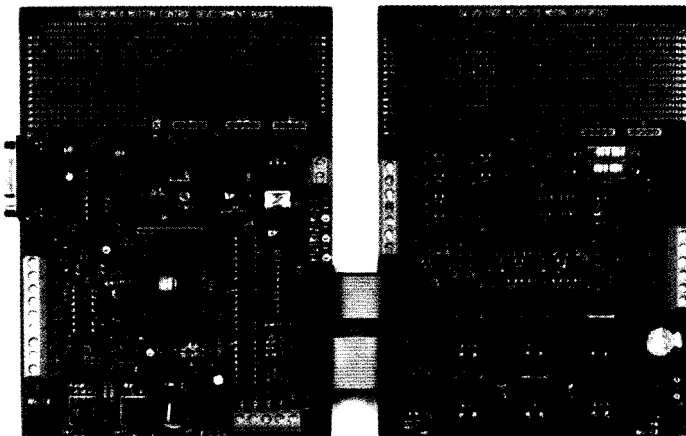
DIGITAL MOTION CONTROL

Motorola's two-board toolset (KITITC127/D and KITITC122/D) simplifies the development of computer-controlled motor drives. The boards can work independently or together to create a motor-drive system for fractional-horsepower DC motors.

A digital motion-control board, based on the MC68HC705MC4 MCU, features all basic motor functions with on/off and forward/reverse switches as well as a speed-control potentiometer. The MCU incorporates a 16-bit timer with an output compare and two input captures, 8-bit ADC with a six-channel input multiplexer, dual-channel pulse width modulator (PWM), SCI, and COP watchdog timer. The 4-KB memory map has 3584 bytes of user-programmable ROM/EPROM and 176 bytes of RAM.

A complete board kit (ITC127/D) includes the motion-control board coded with a basic turn-a-motor program, user app note, key-device datasheets, and two ribbon cables for connection to the power stage (drive and feedback signals). Onboard terminals accommodate Hall-sensor system connections.

The low-voltage power-stage board provides a direct interface between microcomputer-based controllers (e.g., the ITC127) and fractional-horsepower brush and brushless DC motors. It accepts six logic inputs that control three complementary half-bridge outputs. The board also offers current sense, temperature sense, and bus voltage-feedback terminals. The kit (ITC122/D)



includes an app note and datasheets of key components. Space is available onboard for breadboarding user system modifications.

The '705MC4 motion-control development board sells for \$225. The low-voltage power-stage kit costs \$145.

Motorola Customer Response Ctr.
426 N. 44th St., Ste. 150
Phoenix, AZ 85008
(602) 914-8070
Fax: (602) 914-8044
www.mot.com

#501

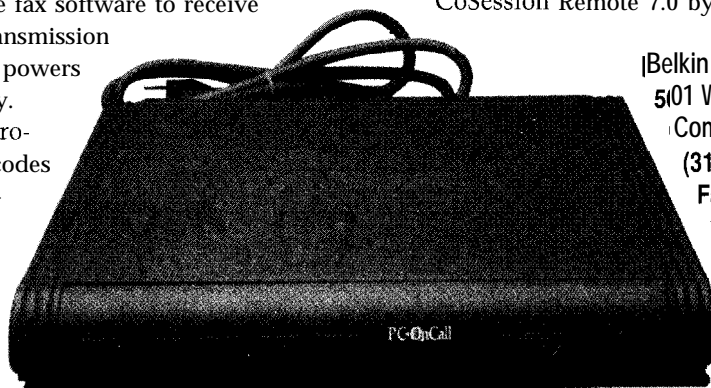
REMOTE-ACCESS POWER CENTER

PC OnCall is an innovative tool that enables users to power up their computer from any touch-tone phone, to transfer files using remote software, or to run their PC remotely. It also detects and properly routes all incoming faxes. Working through a user's phone line or answering machine, the unit monitors the fax, powers up the PC, and loads the fax software to receive the document. After transmission is complete, PC OnCall powers down to conserve energy.

PC OnCall features programmable touch-tone codes to prevent unwanted access. The unit also includes surge-protection technology in a desk-top power-management center equipped

with a heavy-duty 8' power cord, five outlets, a static-discharge plate, and full three-line AC protection rated at 850 J. The product is UL 1449 approved with a rating of 330 V. A \$250 connected-equipment warranty is also offered.

PC OnCall sells for \$129 and comes bundled with CoSession Remote 7.0 by Artisoft.



Belkin Components
5101 W. Walnut St.
Compton, CA 90220-5030
(310) 898-1100
Fax: (310) 898-1111
www.belkin.com

#502

NEW PRODUCT NEWS

LINEAR DISPLACEMENT TRANSDUCER

A microminiature linear displacement transducer featuring a flexible nickel-titanium core has been announced by MicroStrain. The differential variable reluctance transducer (DVRT) is composed of two-layer wound coils, each hermetically sealed inside a 1.5-mm-OD stainless-steel housing with a body length only 2.6 times the linear stroke length. The differential coil arrangement cancels temperature effects and amplifies core displacements. The standard DVRT features 1.5- μ m resolution with filter 3 dB down at 800 Hz, and nonlinearities of 0.30% over 3 mm of stroke.

The DVRT is available with a captive, spring-loaded tip (for gauging); a smooth outer body; or a 400-series SS 4-40-size threaded outer body. Three linear strokes-3, 6, and 9 mm-are currently being produced, and custom stroke lengths are available. Factory calibration and nonlinearity data are shipped with each unit.

MicroStrain, inc.

294 N. Winooski Ave.

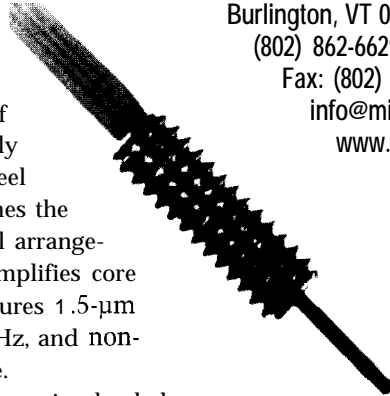
Burlington, VT 05401

(802) 862-6629

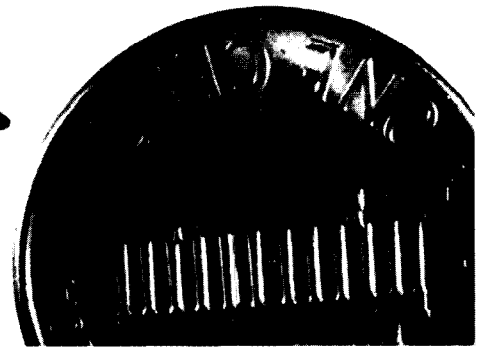
Fax: (802) 863-4093

info@microstrain.com

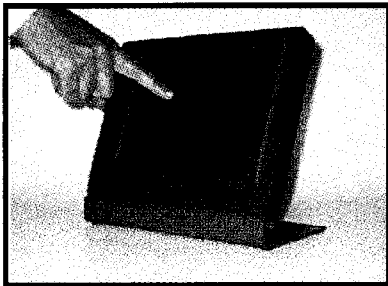
www.microstrain.com



#503



Touch The Future



LCD Touch Monitors

LCD Touch Screens

VGA LCD Displays

LCD Controllers

ISA, PC 104, Analog, Video



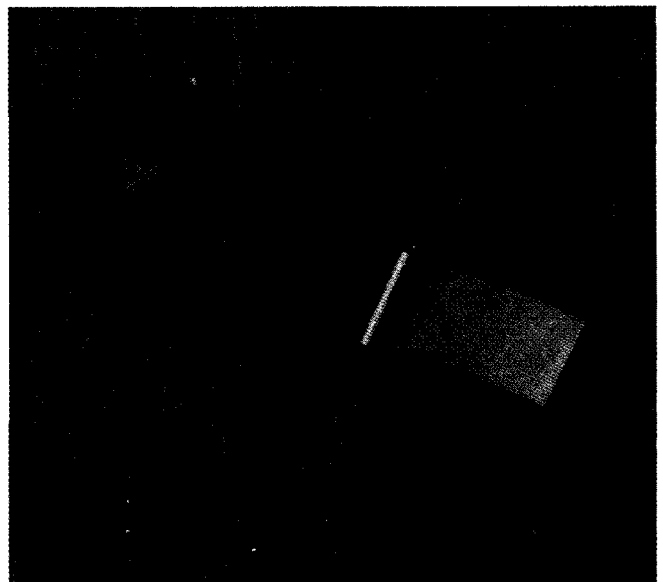
Lowest Prices on Earth!

27101 Aliso Creek Rd - # 154 - Aliso Viejo - CA - 92656

Ph: 714-448-9368 - Fax: 714-448-9316

Email: oemsales@flat-panel.com

FREE CATALOG available at <http://www.flat-panel.com>



E-Series

EPROM - FLASH - SRAM emulation and LIVE editing,
1 to 8Mbit, 70ns access time. Low voltage (3v) options.



Scanlon Design Inc.

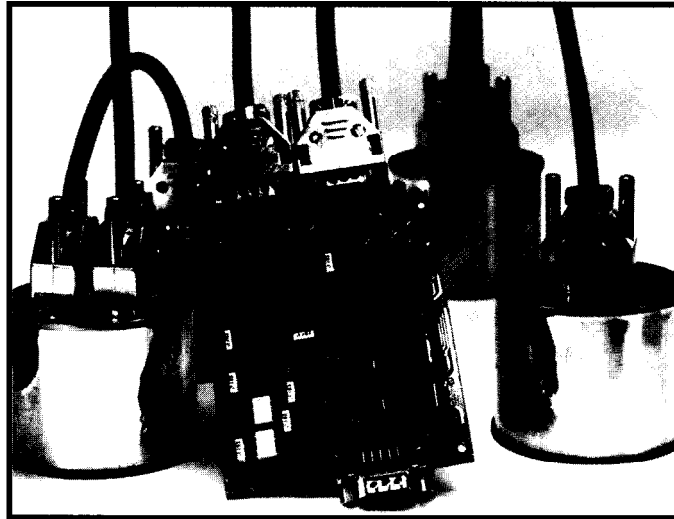
Tel (902) 425 3938 Fax (902) 425 4098
Sales & Info (800) 352 9770

NEW PRODUCT NEWS

CAN-BUS ADAPTER

Distributed real-time control is simplified with the **TDS2020CAN** CAN-Bus Adapter. This multidrop RS-485 serial network, based on the Controller Area Network (CAN), can control relays and lamps in distant equipment, request temperature or other sensors, display messages, synchronize instruments, and query remote dataloggers. Up to 110 nodes can be connected over two twisted-wire pairs (data, and power/ground).

Based on Intel's 82527 CAN-bus IC, the adapter sends data frames on



the serial bus at a 1-Mbps rate as far as 1000 m. Any '2020CAN node can send to up to 14 receivers, whose configuration may be altered by software on-the-fly.

The unit attaches to a '2020 or '9092 Forth controller card to create an intelligent CAN interface. For users familiar with the CAN protocol, all the facilities of

CAN-bus V.2.0 are available (e.g., block memory transfer, text string to remote display, remote read, etc.). For users without CAN experience, high-level software meets 95% of all application requirements for interrupt-driven, optoisolated transmitting or receiving.

The TDS2020CAN sells for \$240.

The Saelig Company
1193 Moseley Rd.
Victor, NY 14564
(716) 425-3753
Fax: (716) 425-3835
saelig@aol.com
www.memo.com/saelig

#504

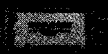
Digital Oscilloscopes

Logic Analyzers



- 2 Channel Digital Oscilloscope
- 100 MSa/s max single shot rate
- 32K samples per channel
- Advanced Triggering
- Only 9 oz and 6.3" x 3.75" x 1.25"
- Small, Lightweight, and Portable
- Parallel Port interface to Laptop or PC
- Advanced Math, TV Line Trigger and FFT Spectrum Analyzer options

\$499



For \$499 you get the model D50-2102 Scope, Probes, Interface Cable, AC Adapter, and Windows and DOS Software.

- 200 MSa/s max single shot rate
- 2 Oscilloscope channels
- 8 Logic Analyzer channels
- 10 Channels simultaneously
- 125 MHz Single Shot Bandwidth
- up to 128K Samples/Channel
- FFT and Spectrum Analyzer

D50-28264 (10Ch, 200MS, 64k) \$1999

D50-28464 (20Ch, 200MS, 64k) \$3799

All prices include Probes and Software

- 40 to 160 channels
- up to 500 MHz
- Variable Threshold
- 8 External Clocks
- 16 Level Triggering
- up to 512K samples/ch

LA4240-32K (200MHz, 40Ch)	\$1350
LA4280-32K (200MHz, 80Ch)	\$2000
LA4540-128K (500MHz, 40Ch)	\$1900
LA4590-128K (500MHz, 50Ch)	\$2800
LA45160-128K (500MHz, 60Ch)	\$7000

All prices include Probes and Software
Call for information on our

100 Year/6 Billion Generation warranty

Link Instruments (201) 808-8990

369 Passaic Ave • Suite 100 • Fairfield, NJ 07004 • Fax (201) 808-8786

Web: <http://www.LinkInstruments.com/cci7> - Email: Sales@LinkInstruments.com

NEW PRODUCT NEWS

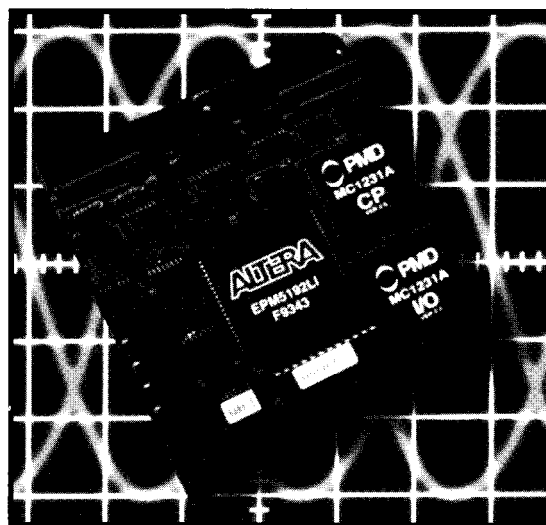
SERVO MOTION CONTROLLER

The **Model 51A Servo Motion Controller** provides two axes of control for IndustryPack-based brushless servo applications that require low electrical noise and low torque ripple at lower motor speeds. With its external sinusoidal commutation capabilities, the Model 51A creates stable systems under conditions that cause standard amplified external commutated systems to fault.

Using PMD's 1231A DSP chipset, the Model 51A provides tighter control of motion by allowing the host PC to manage onboard commutation. The 1231 handles servo algorithms using a PID with velocity feed-forward filtering for each axis and also offers velocity phase advance capabilities. Board initialization can be Hall based or algorithmic.

Software libraries for the Model 51A are compatible with most C, C++, Visual Basic, Visual C++, and Turbo Pascal compilers. These libraries also include Windows DLLs. Software is available which enables motion to be coordinated between any two axes on identical boards sharing the same backplane.

The Model 51A sells for \$995. The **Model 51A Development Kit** provides all hardware, manuals, and software libraries for \$495.



Technology 80, Inc.
658 Mendelssohn Ave. N
Minneapolis, MN 55247
(612) 542-9545
Fax: (612) 542-9785
www.tech80.com

#505

TIRED OF WAITING FOR THE PROMPT ?

Speed up with a ROM DRIVE! Boots DOS programs instantly. Also used to replace mechanical drive completely in controllers or diskless workstations. The only perfect protection from viruses. Easy to install half-size card.

MVDISK1 128k \$75
MVDISK2 144m \$150
MVDISK3 768m \$265

\$75

Quantity discounts!

DOS IN ROM!



\$95 EPROM PROGRAMMER

- Super Fast Programming
- Easier to use than others
- Does 2764/27080 (8 Meg)

WORLD'S SMALLEST PC !!!

ROBOTS ALARMS RECORDERS DOS

THREE EASY STEPS: **\$27** 1K QTY
1. Develop on PC
2. Download to SBC **\$95**
3. Burn into EPROM **\$95** SGL QTY

-2 PARALLEL -LCD INTERFACE
-3 SERIAL -KEYBOARD INPUT
-PC TYPE BUS -REAL TIME CLK
-BIOS OPTION -BATTERY OR SV

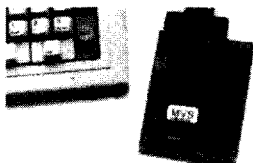
FREE SHIPPING IN U.S.

5 YEAR LIMITED WARRANTY

MVS

Box 850
Merrimack, NH
(603) 782 8507

8088 SINGLE BOARD COMPUTER



JUMP START YOUR DESIGNS with these practical guides

NEW! "A focused book that delivers what it promises: detailed technical information on the parallel port." - *Windows Developer's Journal*

"It's been a while since I've seen a book as practical as this one."
- *Nuts & Volts*

ISBN 0-9650819-1-5

\$39.95
343 pages
Includes disk

Parallel Port Complete

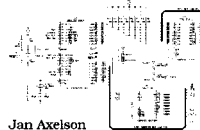
Programming, Interfacing, & Using the PC's Parallel Printer Port

- ✓ Includes EPROM
- ✓ IEEE-1284
- ✓ Source code in Visual Basic
- ✓ User tips

Jan Axelsson

The Microcontroller Idea Book

Circuits, Programs & Applications featuring the 8052-BASIC Single-chip Computer



Jan Axelsson

MICROCONTROLLERS!

"An ideal introduction to low-end embedded design." - *EDN*

"The writing is a model of clarity and conciseness." - *PC Magazine*

ISBN 0-9650819-0-7
273 pages. \$31.95

Order line: **1-800-247-6553**

For more information or international orders:
Lakeview Research Phone 608-241-5824
2209 Winnebago St. Fax 608-241-5848
Madison, WI 53704 Email jaxelson@lvr.com

Shipping: \$5.00 (U.S. & Canada)

Read sample chapters on the web:

<http://www.lvr.com>

FEATURES

12

Genetic Algorithms for FPGAs

18

When 64 KB Isn't Enough

24

Windows-based Development System for the 68HC705C8

62

Test Drive a Precision Motion Controller

Genetic Algorithms for FPGAs

The Borg are here—at least in code! Ingo defines genetic programming and shows how it emulates the evolution of biological organisms as they adapt to environmental changes.

FEATURE ARTICLE

Ingo Cyliax



When I was first exposed to genetic programming, I was working on robot controllers at Indiana University and was skeptical about its merits.

Now that I've seen robot gaits generated with genetic programming, I'm realizing that genetic programming has some applications and that it's not all that mystical.

Genetic programming has been around for more than three decades and has solved problems in areas such as economics, biochemistry, and engineering.

Researchers at our lab are trying to use genetic programming to program behaviors such as walking gaits in six-legged robots (see the Stiquito on the cover of INK 81 as well as "Modular Robot Controllers," INK 73).

Genetic programming attempts to emulate how biological organisms evolve to adapt themselves to environmental challenges. In evolution, genetic information, which acts as the blueprint for the organism, changes from generation to generation in an effort to find different ways to survive.

In biology, techniques for this are fairly diverse. Luckily, genetic programming is a simple abstraction.

Several different algorithms have been developed to emulate or approxi-

```
A: 0 1 2 3 4 5...
B:  . . . . . 6 7 8 9
Result: 0 1 2 3 4 5 6 7 8 9
```

Figure 1—In this example, the crossover point was chosen to be 6. The resultant genome contains bits 0-5 from genome A and bits 6-9 from genome B.

Member	Fitness
1	10
2	7
3	5
4	5
5	1

Probability Distribution:
[1111111111][2222222][33333][44444][5]

Table 1--The selection process tries to pick an individual from the population based on their fitness. Here, the fitness values range from 1 (worst) to 10 (best) for each member. Member 2 has a one-in-four chance of being picked.

mate this process in software. In this article, I describe one of the most common algorithms and discuss how it might be adapted for more specific applications.

In a nutshell, a genetic algorithm (GA) operates on genomes--the basic information carrier. The collection of genomes is called the population.

In each step (called a generation), the population is evaluated and a qualitative index is assigned to each genome. This index is called the fitness, and the evaluation is called the fitness function.

The genomes are then ranked, and pairs of genomes are selected to generate an offspring for the next generation. This process repeats for many generations until the particular solution is found or some fixed number of generations elapses.

Let's look at the algorithm in detail.

The genome is the data structure that contains the information necessary to implement or solve a problem. The encoding of the bits in the genome is only important during evaluation of the fitness function.

The rest of the GA treats the genome as a generic fixed-length stream of bits. In practice, the length can vary between tens of bits to over 1000, depending on the problem.

By itself, a single genome (i.e., an individual) is not very useful in a GA. There's no mechanism to quickly change the contents of the genome. We need to have several individuals (i.e., a population).

Large populations are good because they can contain much variety. However, they take longer to evaluate.

Each generation operates on one population and evolves a new popula-

tion based on the old one. The old generation is then forgotten.

The first population needs to be initialized, which can be done in many ways depending on the problem. The most common method is to fill it with random numbers.

If the programmers have some idea of what the outcome may be, they can try to initialize it with some a priori knowledge to bootstrap it. In many cases, however, it's probably better to start off random, since the solution may not be obvious. And, a bad guess may hurt the performance.

At the beginning of each generation, all the genomes need to be evaluated. Unless the genome actually represents the information being searched for, a simulation model for the system needs to be run which uses the genome as parameters.

For practical systems, the model may be very complex and take most of the GA's computing resources. Some simulations may run in several domains (e.g., thermo-electromechanical systems). It's these complex systems, which are hard to solve using traditional engineering techniques, that apply themselves well to genetic programming.

After each simulation runs, the results are evaluated by a function that

estimates the quality of the genome from the simulation. This quality, called the fitness, is used to rank each genome from best to worst. Typically, the fitness function is complex, since it may have to evaluate several merits of the system to arrive a global "goodness" factor.

To generate the next generation, two individuals from the current population are selected. The members are chosen randomly, but the probability has to be relative to their fitness. Table 1 shows an area distribution of the fitness.

Once two members are chosen, a bit position for the crossover is selected at random. The crossover point defines how many bits are taken from each of the parent members to generate the offspring.

The crossover is what gives a GA its dynamic features, and there are many variants of the simple function I present here. Figure 1 shows what the crossover looks like.

In many cases, the GA may only be able to find a local maxima in the solution space and then get stuck. Introducing a mutation (i.e., random bit flips) forces the GA to consider other solutions.

To best illustrate this algorithm, let's look at a trivial example of de-

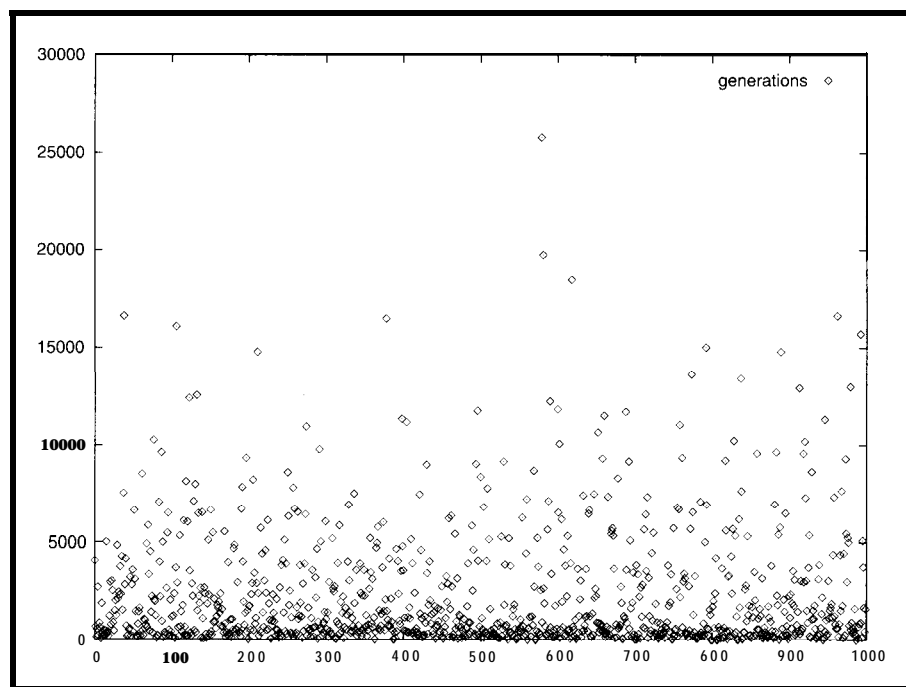


Figure P--This figure shows the distribution of generations it took to evolve the mod-8 counter over 1000 trials. On average, it was able to find the counter in ~2500 generations. Even the worst trial, at ~26,000 generations, was still better than a brute-force search of 2^{24} entries.

Original Population	Fitness
01 01 1000	2
01 01 1001	1
10101100	0

Individuals Chosen	Crossover Bit	New Population (with Mutation)
01 01 1000 01 001001	7	01 01 1001
01 01 1000 01 01 1000	2	01 01 1000
01 01 1000 0101 1000	5	0101 1000
01 001001 01 001001		01 00 10 (1)1

New Population	Fitness
01 01 1000	2
01 01 1000	2
01 001000	2
01 001000	2

Table 2—One generation involves ranking individuals based on fitness and selecting two individuals for crossover resulting in mutation in the new population

signing a mod-4 counter. A mod-4 counter has 4 states and can be encoded in 2 bits.

Let's represent the genome as a string of four 2-bit digits which can contain the state encoding at each of the states. For the counter to work properly, the state encoding must look like: "00 01 10 11" (i.e., it counts 0, 1, 2, 3)

The fitness function counts how many of the 2-bit cells are in their correct position. The correct solution has a fitness of "4."

Table 2 shows one generation of this GA. A population of four genomes is evaluated for fitness and ranked (2 1 0 0). Then, members are chosen based on their fitness distribution.

The best member has a two-thirds and the second a one-third chance of being chosen. We also randomly select four crossover points (7 1 2 5) and perform the crossover function to generate four offspring genomes.

A one-bit mutation is indicated in parentheses. When we evaluate the fitnesses of the new generation, we discover that even though we have lost the best one (fitness of 3), the average fitness has improved from 0.75 to 2.0.

This example was simple. There are only $1/256$ correct choices, and doing a brute-force search of the solution space would have been easy.

However, if I increase the complexity of the problem by implementing a mod-8 counter, the genome is now 24 bits (eight 3-bit digits) long and the

solution space is 2^{24} (~16 million), of which only one solution is entirely correct.

I wrote a program to try to find a mod-8 counter using the GA described here. Out of 1000 trial runs, it took anywhere from 50 to 26,000 generations with a population of 16 genomes. The average was 2300 generations (see Figure 2 for a distribution).

This result is much better than would be possible with a brute-force search. In Figure 3, you can see the dynamics of a single trial of about 250 generations.

In both examples, I assumed the genome represents the information I'm looking for. In real life, the genome may represent a parameter that needs to be decoded to be useful.

You may know what the appropriate behavior of your system is, but the relationship between the parameter space and the system's behavior is most likely nonlinear and complex. Otherwise, the solution would be easy.

Let's look at some systems which can be solved by letting the genome represent a more complex parameter.

GAs can be used to find filter parameters for complex filters or even systems of filters. In this case, the genome may represent critical filter coefficients, and the fitness function would evaluate a filter based on the desired performance.

The performance can be the phase/gain relationship of the filter but can also include things like heat dissipation, component tolerances, and so forth.

In our lab, one graduate student used a variant of GA called cyclic genetic algorithm (CGA) to evolve gaits for hexapod robots.

In a CGA, the genome consists of a string of genes that represent the leg-actuation pattern and a duration. These genes are strung together and, at some

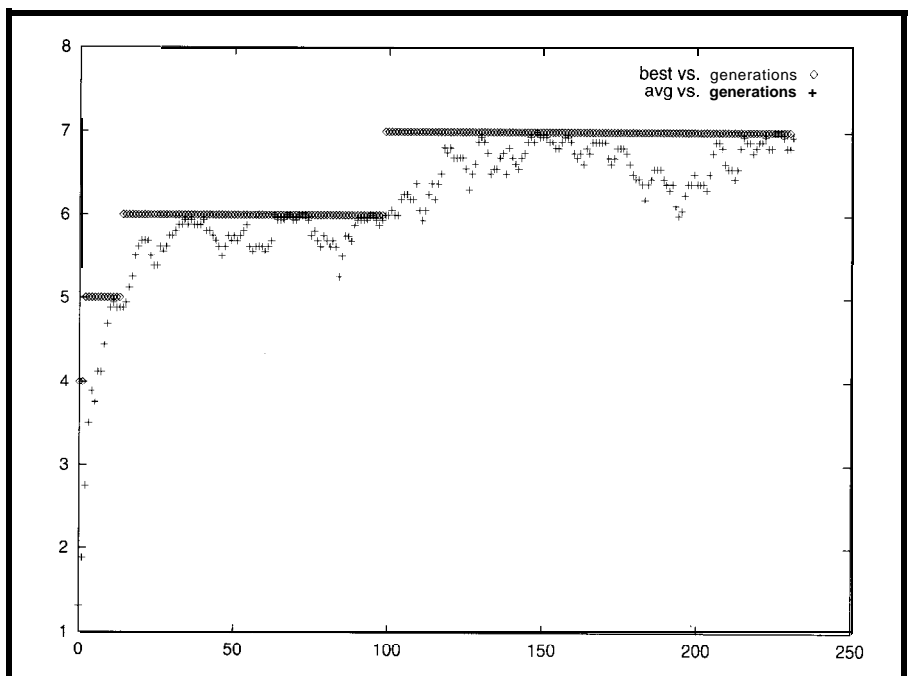
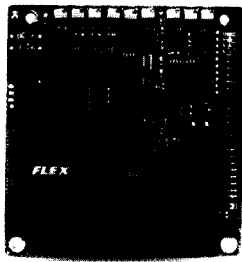


Figure 3—This trial of 250 generations shows an ideal case, where the genetic algorithm approaches the correct answer roughly asymptotically.



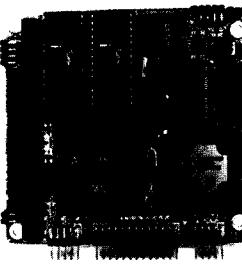
Sets The Pace In PC/104 Data Acquisition

Scan 16 Channels...
Any Sequence...
Any Gain...
500 kHz



DM6420 500 kHz Analog I/O Module
with Channel-Gain Table and FIFO

With Companion
Am5x86™ 133 MHz
PC/I 04 cpuModules



The CMV586DX133 offers
versatile embedded functionality

Our PC/104 and ISA Bus
Product Lines Feature
Intelligent DAS Cards With
Embedded PC and DSP,
Analog and Digital I/O, CPU,
Shared Memory, SVGA, PCMCIA,
CAN Bus and GPS Modules

Real Time Devices USA Inc.
200 Innovation Boulevard
State College, PA 16804-0906 USA
Tel:1(814) 234-8087 • Fax:1(814) 234-5218
URL:www.rtdusa.com
E-Mail:sales@rtdusa.com

RTD Scandinavia Oy
Helsinki, Finland
Tel:358-9-346-4538
Fax: 358-9-346-4539

RTD is a founder of the PC1104 Consortium

point, form a loop to cycle through several leg-actuation patterns.

The CGA is an example of a variable-length GA, since the number of actuations can vary for genomes. The CGA selects the leg-actuation pattern to use, the duration, how many actuations there are, and at which point the pattern starts repeating (looping).

The evolved genomes are implemented as state machines which are synthesized into a controller FPGA. The FPGA then controls the actuators of the robot directly.

In tests, the CGA was able to come up with better (faster) gaits than my hand-coded gaits. By the way, the student is now working on evolving eight-legged gaits for octopods, which is interesting since octopods can have several walking modes.

You can also have the GA directly generate FPGA configurations. This task requires the use of special FPGAs, since many of the common FPGAs use internal tristate buffers that can cause fights when not configured correctly.

When doing genetic programming for FPGA configurations, it's a nice feature not to have your FPGAs burn up when the GA ends up trying a genome which should have just received a low fitness ranking.

Up to now, I've assumed the GA used to generate the genomes is implemented as a program running on a computer. The program performs the fitness evaluation, crossover, and mutation until a genome is found which may be suitable for implementation in system.

Another strategy is to implement some-or maybe even all-of the functions of the GA in hardware. The crossover and mutation functions are just bit operations, and if the fitness function can also be implemented directly in hardware, it will speed up the search dramatically.

Check out some of the Web resources cited at the end of the article to find out what's being done with GAs in hardware. This field is also referred to as "evolvable hardware."

INTO THE FUTURE

Traditional GAs are certainly an interesting way of programming where

the computer does most of the work of finding solutions to complex problems.

However, with the arrival of suitable FPGAs, we'll see GA applications that use these devices to dynamically reprogram themselves to adapt to changes in their environment or cope with failures.

I'm not ready to trust such systems quite yet. Perhaps this has to do with the mostly negative examples of such systems in the science-fiction literature, where they tend evolve into evil adversaries like HAL9000. I'm hoping that real evolvable-hardware systems will be more benign. □

Ingo Cyliax is a research engineer in the Analog VLSI and Robotics Lab and teaches hardware design in the computer science department at Indiana University. He also does software and hardware development with Derivation Systems, a San Diego-based formal-synthesis company. You may reach Ingo at cyliax@EZComm.com.

SOFTWARE

The genetic program to find the mod-8 counter mentioned in the article can be found at ftp.cs.indiana.edu/pub/goo/GA/gacnt.c.

REFERENCES

Internet

www.cs.indiana.edu/hyplan/gaparker.html
www.cogs.susx.ac.uk/users/adrianth/index.html
lslwww.epfl.ch/~moshes/firefly.html
splash.ee.byu.edu

Texts

D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
J.H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, MI, 1975.

401 Very Useful
402 Moderately Useful
403 Not Useful

FEATURE ARTICLE

Jim Sibigroth

When 64 KB Isn't Enough

Remember when a development system used only 4 KB and no software tools could manage more than 64-KB software projects? Obviously, those days are gone. Jim suggests how to add memory by using a bank-switching system.



When microprocessors were first introduced around 1974, 64 KB of memory seemed like more than anyone could ever use. A full-blown development system from a major IC manufacturer had plug-in boards with only 4 KB of dynamic RAM.

There were no commercial software tools that could manage a 64-KB software project—even if someone *did* have the patience to write that much assembly language. Needless to say, times have changed.

Today, even a modest embedded control application runs past the 64-KB boundary. This is especially true if you use a high-level language like C.

Typical midrange microprocessor architectures have a 64-KB memory space limit due to their 16-bit address bus. In such a system, you have two primary options. You can switch to a different microprocessor with a wider address bus, or you can implement a bank-switching memory system.

On the surface, an MCU with a wider address bus sounds pretty good. But, consider the costs. Typically, changing processors involves an expensive learning curve to rework old software for the new CPU.

Also consider that processors with large linear address space take more

bits to address a particular location. So, instructions—and ultimately, programs—take more memory space.

While bank-switching systems typically take less program memory space, they are not without their own drawbacks. Programs need to be broken into blocks no larger than a single bank (e.g., 16 KB). Usually, extra programming is needed to change from one page to another.

In this article, I explain how you can implement a bank-switching system on almost any MCU.

However, the Motorola MC68HC812A4 includes a similar system with some interesting enhancements that greatly simplify the use of a bank-switching system. So, I also explain some limitations of traditional bank-switching systems and show how the 'HC 12 overcomes them.

If you're not familiar with bank switching, take a look at the sidebar, "Common Bank-Switching Terms."

BANK-SELECT MEMORY

Figure 1 shows the logic needed to implement a bank-switching system for a 4-MB physical-memory and 16-KB expansion-window size. Its logic can be implemented in a programmable logic device, but for clarity, it's implemented here in simple HCMOS logic devices.

Figure 1 helps explain the address generation and multiplex logic needed for bank-select memory. After seeing how this logic works, you should be able to extend the idea to other types of memory, including RAM.

The MC68HC812A4 has a 16-KB program expansion window that's functionally the same as this system, except the 'HC12 has a full 16-bit data bus. In addition, the MC68HC812A4 has two other expansion windows—a 4-KB data-expansion window from \$7000 to \$7FFF and a 1-KB "extra" expansion window that can be located at \$0400–\$07FF or \$0000–\$03FF.

The block size in a bank-switching system is typically some power of two [e.g., 1K, 4K, 8K, 16K, or 32K]. However, it isn't normally 64K because that leaves none of the 64-KB space for unpagged memory (i.e., common or resident memory space).

The banked portion of memory is viewed by the processor, one bank at a time. Some resources (e.g., on-chip control registers, RAM for a stack, and interrupt vectors) need to be accessible by the CPU at all times (regardless of which bank is currently selected).

If the banks take all 64K, vectors need to be duplicated in every bank, which isn't an efficient use of memory. If the banks are too small, you spend too much time switching between them.

A 16-KB program window is relatively large and doesn't interfere with the vector space or on-chip resources (e.g., control registers and RAM). The unpagged spaces also leave plenty of room for a large contiguous block of system RAM and a 16-KB block of resident program memory at \$C000-\$FFFF, which includes the vectors.

In Figure 1, a 2-4-line decoder (1/2 'HC139) divides the 64-KB memory map into four 16-KB areas. Output Y2a drives low whenever the CPU address is in the area \$8000-\$BFFF, which is the expansion window for the bank-switched memory.

When the CPU address is outside this window, the upper 'HC244 is

enabled and CPU address lines A15 and A14 pass through to the external memory system. The other six inputs to the 'HC244 are tied to V_{dd} so expansion address lines XA21-XA16 are forced to 1s.

When the CPU address is within the expansion window, the upper 'HC244 is off and the lower one is enabled. The lower 'HC244 passes the current value in the PPAGE latch ('HC373) to the expansion address lines XA21-XA14.

The PPAGE latch looks like an 8-bit control register to software. Its value determines which one of the 256 pages the CPU sees in the \$8000-\$BFFF address space. Although Figure 1 doesn't show how to read PPAGE, some code in this article assumes it can be read.

Jumper J1 has two ways to enable the external memory system. If the jumper is shorted across pins 1 and 2, the Y2a output of the 'HC139 drives the memory system's chip select, causing the external memory to appear only within the bank-select window from \$8000 to \$BFFF.

When J1 has a short from pin 2 to 3, inverted CPU address line A15 drives the chip select for the memory system.

Although this feature introduces an addressing ambiguity, it's useful since it enables a single external memory device to include the unpagged area from \$C000-\$FFFF (contains the reset and interrupt vectors) and up to 256 banks of 16 KB each.

The addressing ambiguity arises because two different logical CPU addresses can access the highest 16-KB block in the 4-MB physical memory. The CPU address \$FFFF is not within the bank window, so the top 'HC244 is enabled, producing a physical address of \$3FFFFFF at the memory system.

The address \$BFFF in the last bank (PPAGE = \$FF) is in the bank window, so the lower 'HC244 passes the PPAGE value (\$FF) to XA21-XA14. This also produces a physical address of \$3FFFFFF at the memory system.

The result is an interesting tradeoff possibility in the 256th 16-KB bank. You can choose to allocate a portion of this 16-KB block for unpagged vectors and routines that are always accessible to the CPU. The remaining portion of this last page can be used as a partial banked page or as additional unpagged space.

Just don't try to fill the 256th bank and write other code that goes at \$C000-\$FFFF. There really is only 16 KB of physical memory. (The two logical areas are the same 16-KB physical-memory location.)

In an 'HC12 system, this ambiguity is exploited to allow a single external EPROM for vectors and unpagged memory at \$C000-\$FFFF. And, the rest of the EPROM can be used for several 16-KB banks (depending on the size of the external EPROM).

This option is less expensive than using separate devices for pagged and unpagged memory space.

Figure 2 shows the memory map for the system shown in Figure 1. Logical CPU addresses are shown on the left.

In an MC68HC11, on-chip RAM is typically located at \$1000 and the on-chip registers are located at \$0000. The on-chip RAM and/or registers can be remapped to any 4-KB boundary by writing a value to a control register.

The usual purpose for this is to enable the user to place the most frequently accessed resource in direct

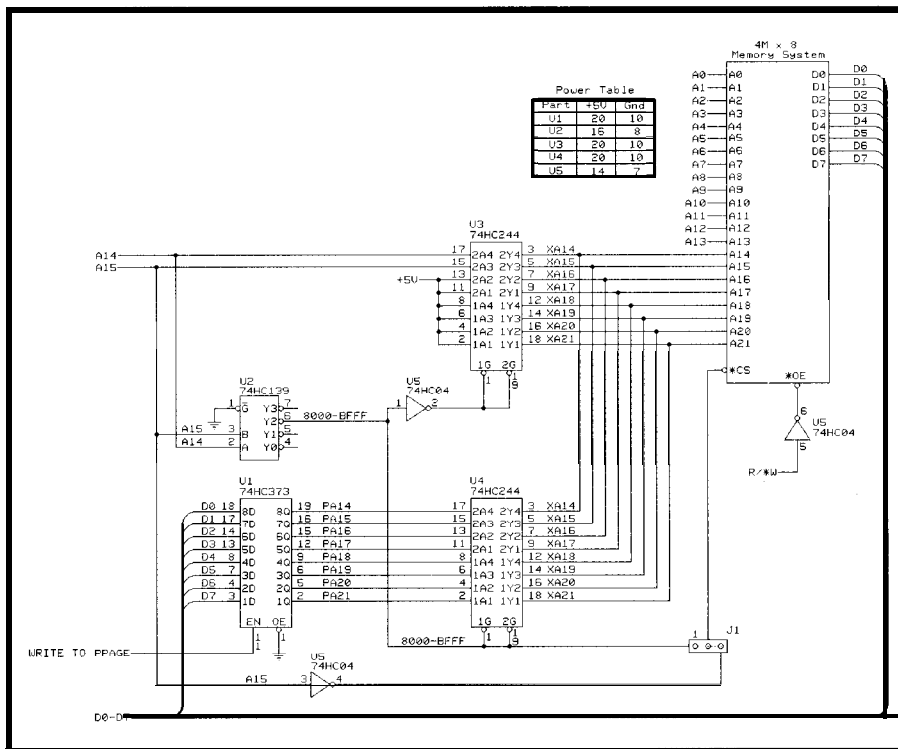


Figure 1—This schematic is suitable for any 8-M microprocessor with a 16-bit address bus (e.g., an MC68HC11F1). The 'HC373 holds the page number of the currently active bank. The 'HC244 buffers form a 2:1 mux whose output drives the high-order address lines XA[21:14]. J1 determines whether the memory system includes resident memory at \$C000-\$FFFF as well as banked memory in a window at \$8000-\$BFFF or just banked memory.

memory space (\$0000-\$00FF). But, you can remap them to the WAGE window in this example system (\$8000-\$BFFF).

On-chip resources have priority over any external access, so if the on-chip RAM or registers are remapped to \$8000, they appear to be duplicated on all banked pages. Although this is interesting, it isn't normally done because part of each bank of the external memory becomes inaccessible.

The external memory system is shown as a series of 256 banks or pages of 16 KB each. The physical address range for each page is shown along the right edge.

The CPU always generates an address in the \$8000-\$BFFF range for every page of banked memory. The current value (page number) in the WAGE register at any given time determines which page is accessed.

CPU address range \$C000-\$FFFF corresponds to expansion addresses \$3FC000-\$3FFFFFF. Notice that this physical address range is the same as page FF of the banked memory.

PROGRAMMING PAGED MEMORY

This code shows how you typically jump to an arbitrary location in the banked memory:

```
LDAA  dest_page
STAA  PPAGE
JMP   dest_addr
```

Listing 1—Calling a task subroutine in a remote bank, the first portion of this code may be anywhere in memory, including within a bank of extended memory. The second part must not be in paged memory space

```
;code in mainline routine where the remote task is called from
LDAA  dest_page ;destination page #
LDX   dest_addr ;destination address
JSR   task-call ;common calling routine

;common calling routine in unpagged memory
task-call:
LDAB  PPAGE     ;old page #
PSHB  ;save on stack
STAA  PPAGE     ;change to new bank
JSR   0.X       ;call task
PULA  ;recover old bank #
STAA  PPAGE     ;change to old bank
RTS    ;return to mainline
```

Dest_page is the bank number (\$00-\$FF) where the destination is located. Dest_addr is a CPU address in \$8000-\$BFFF where you want to jump.

But of course, things aren't that simple. For starters, this sequence doesn't work unless it's located outside the bank window. If you try this from within one bank and try to jump to another, the CPU gets confused between the second and third instructions.

As soon as PPAGE is written to the new value, the old bank is replaced by another. So, JMP is gone when the CPU tries to execute it. One solution is to ensure all page-changing operations are located in resident (unpagged) memory.

Fortunately, it isn't common to jump from one bank to another—with one

possible exception. Some program segments may be too large to fit in a single bank. Then, you must jump to a page-increment routine (in unpagged memory) just before the end of the current bank.

The page_adv routine can be a single common routine for going from the end of any bank to the beginning of the next consecutive bank:

```
page_adv:
INC  PPAGE
JMP  $8000
```

It must be located in unpagged memory (e.g., in the \$C000-\$FFFF area). Even counting a JMP page_adv near the end of the previous page, this sequence is still very small and fast.

Common Bank-Switching Terms

Bank (or Page)—a block of memory that can be accessed by the CPU. In a bank-switched memory system, a large physical memory is conceptually broken into a number of logical banks or pages that can be switched into the memory map of the microprocessor, one bank at a time. The maximum size is limited only by the number of bits you choose to use in the bank-select logic. An 8-bit page number allows for 256 banks.

Page Register—a control register where the bank number of the currently visible bank number is stored. To switch in a different bank, a different page number is written to the page register.

Physical Address—an address within a physical memory chip or system. For example, a 1-Mb EPROM has 128 KB, so physical addresses in this memory range from \$00000 through \$1FFFF. A microprocessor with 16 address lines can only address 64 KB, so it can't directly address all of such a large EPROM.

Logical Address—the address of a memory location from the CPU's point of view. For a typical midrange CPU, such an address includes an address within the processor's 64-KB memory map and a bank number.

Address Multiplexer—circuitry that combines the CPU's 64-KB address with the location's bank number to form the physical address of a specific memory location. Its output provides the highest order address lines to the memory system.

Expansion Window—a range of addresses in the memory map of the microprocessor system through which banks are viewed by the CPU. At any particular time, only one bank is accessible through this window.

Common (or Resident) Memory—the portion of the 64-KB memory space of the CPU that isn't within any expansion window. Such memory is always accessible by the CPU regardless of which bank is selected.

The more common program structure in a banked-memory system places the mainline program in unpagged memory and calls (using J S R instructions) other task routines, which are each completely contained within a single bank. With this structure, you only need to change pages at the start of each major task.

J S R brings up another problem with bank-switching systems. J S R tries to remember the source address by saving it on the stack.

But, PPAGE is also part of the physical address for the source. Therefore, you should save the old PPAGE value before J S R and restore it on return.

One way to code it into a single calling subroutine is to load the desired destination page number into an accumulator and the destination CPU address into an index register before calling a task-switching routine (see Listing 1).

The task-switching routine (located in unpagged memory) then calls the remote task. It only appears once in an application, but LDAA and LDX are needed for each call to a remote task.

In Listing 1, all PPAGE changes are done using instructions located in unpagged memory. This sequence also uses up the A accumulator and an index register.

While this creates significantly more overhead than a simple J S R, it still may be better than changing to a more expensive processor. This extra overhead to swap pages is the biggest objection to bank-switched systems.

The MC68HC812A4 greatly simplifies this process by including two new instructions-CALL and RTC (return from call). They work much like J S R and RTS except that CALL also stacks the old PPAGE value and changes PPAGE to the desired destination value, and RTC restores the old PPAGE value as well as the program counter.

In the 'HC12, you simply write:

```
CALL dest_addr, dest_page
```

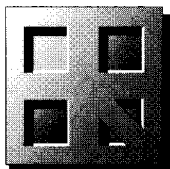
CAL L instructions don't have to be located in unpagged memory. You can CALL from one bank to another and return using RT C with no undesirable side effects.

It's a very simple formula.
Buy a PromICE from Grammar Engine and you'll save money **3** ways. First, PromICE is amazingly

We figured out a way to save you money while you're developing!

affordable, starting at just \$495. Then, you can re-use PromICE on project after project because it works with any micro, allowing you to avoid high-cost custom tools. Finally, there is the investment you've made in a tool that is easily upgradeable as your needs change. Call us today and try PromICE FREE for **30** days. It may just be the best investment you can make in firmware development tools.

1-800-PROMICE



Grammar Engine, Inc.

921 Eastwind Dr., Suite 122 • Westerville, OH 43081
Phone 614/899-7878 • Fax 614/899-7888 • Sales 800/776-6423
email: info@gei.com • Web: http://www.gei.com

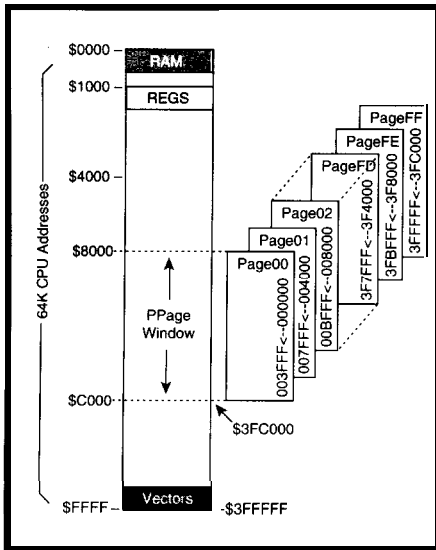


Figure 2—This memory map shows the CPU's 64K addresses along the left side. The \$8000-\$BFFF space is a window through which a large 4-MB space can be viewed, one 16-KB page at a time.

All the information needed to complete the entire PPAGE switch is in the CPU before PPAGE actually changes. This feature eliminates the danger of disabling the source bank in the middle of executing the code that performs the switch sequence.

Routines called with JSR end with RTS, and those called with CALL end with RTC. You can use CALL and RTC even if the subroutine is in unpagged memory or in the same bank as the CALL instruction, but it's less efficient than using JSR and RTS.

If a subroutine is located in one bank and sometimes called from another, it must end with RTC. Therefore, it must be called with a CALL instruction even for calls from within the same page.

Interrupts can also cause problems for bank-switched systems. The concept of an interrupt is that the preinterrupt context is saved on the stack so it can be restored after the service routine finishes.

The PPAGE register in a bank-switched system is part of the context. But, it isn't a CPU register and therefore is not automatically saved when an interrupt occurs. In the bank-switched system of Figure 1, the interrupt vectors are in unpagged space, so they're always visible to the CPU.

The start of the service routine must also be in unpagged memory. This way, the CPU can start executing it

without first modifying the PPAGE register.

The main body can be in some other bank as long as the service routine saves and restores the original PPAGE value (using instructions in unpagged memory). This technique allows code to resume properly after the return from interrupt (RTI).

In the 'HC12, CALL and RTC automatically handle the page swap and restore. So, an interrupt service routine can consist of a CALL followed by an RTI in unpagged memory. Such code enables the bulk of the service routine to be located in any bank of expansion memory.

Stack memory is also normally located in unpagged resident memory. However, with a lot of extra care, a separate paged memory (e.g., DPAGE in the 'HC12) can be used for the stack.

If the stack is in paged memory and the bank page is changed, there needs to be a mechanism to save the old page number or the stack fails.

One solution is to save the old stack pointer and page number as the first items in the new stack. When it's time to return to the old stack, the bank page-select register and stack pointer can be restored to these values.

Be careful to block all interrupts while changing the DPAGE bank. Otherwise, an interrupt can occur halfway through the sequence and cause information to be stacked to inappropriate physical locations.

PAGED MEMORY FOR DATA

Bank-switched memory systems can also be useful for data. Bank switches are much easier to manage when you're not trying to execute code from within the bank window. For a large data structure, set the page register to the desired bank and then access the desired data.

In such a system, you should avoid defining data structures where a data record can straddle bank boundaries. That makes it messy to access all the data in the record (i.e., you have to change the page register in the middle of the operation).

A data logger is an application that might use bank-switched memory to log data. When one bank becomes full,

the program simply switches to the next bank.

The MC68HC812A4 has a 16-KB program window (PPAGE), a 4-KB data window (DPAGE), and a 1-KB extra window (EPAGE). CALL and RTC only work with PPAGE.

DPAGE and EPAGE are traditional bank-switching systems intended primarily for data. You can put programs in them, but you need to use traditional bank-swapping techniques instead of the more efficient CALL and RTC.

GET WITH THE PROGRAM

Bank-switched systems can cause extra challenges for programmers. But, they typically result in smaller program size than systems with a wider address bus.

Since programming is a one-time engineering cost and a larger ROM makes every end product more expensive, the bank-switching system is often a better choice. The CALL and RTC instructions in the MC68HC812A4 greatly reduce the problems associated with bank switching by incorporating the entire page-changing operation within an uninterruptible instruction. ■

Jim Sibigroth is a system design engineer working on advanced microcontrollers for Motorola. His latest project was the '68HC12, where he was a coarchitect for the CPU12 instruction set. He devised the memory-expansion system described in this article. You may reach him at jims@seasick.sps.mot.com.

SOURCE

MC68HC812A4

Motorola
MCU Information Line
P.O. Box 13026
Austin, TX 78711-3026
(512) 328-2268, x950
Fax: (512) 891-4465
freeware.aus.sps.mot.com

IRS

404 Very Useful
405 Moderately Useful
406 Not Useful

FEATURE ARTICLE

Brian Millier

Windows-based Development System for the 68HC705C8

The advent of faster and fancier chips has not brought a similarly enhanced user interface. And, in the low end, although development tools are fine, they run far too slowly. Brian to the rescue. He offers a development environment that really clocks.

am I alone in thinking that electronics and computers are getting fancier and faster but not necessarily easier to use?

I recently replaced a PC sound card, and in the process, I went from a board with a few jumpers and a short, functional setup to one with no jumpers and a setup that loaded a megabyte of files onto my hard disk just for the install!

The same trends seem to be occurring in the microcontroller arena. After 25 years of technological change, we've made little progress in simplifying microcontroller development for the small user.

I'm a big fan of the K1 and J1A devices in the

Motorola 68HC705 family. But while their low-cost development boards are helpful tools, they emulate the device at about $\frac{1}{30}$ of real-time speeds.

In this article, I present a development environment that addresses this problem for the Motorola 68HC705C8.

FINDING MR. RIGHT

Matching the best possible microcontroller and development system for a small user designing embedded controllers requires some investigation.

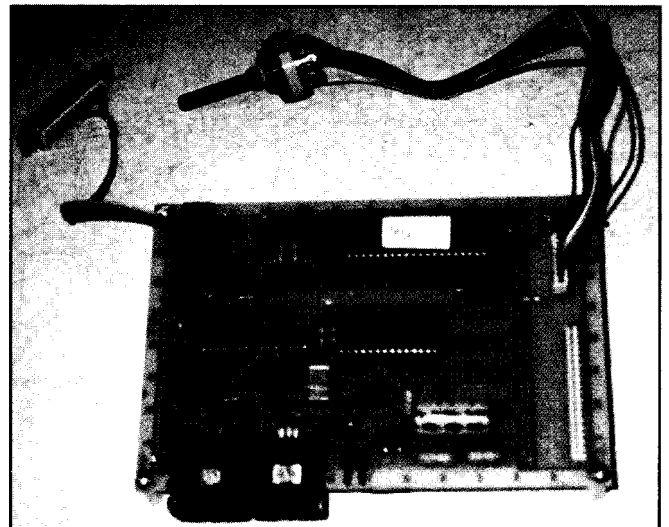
While I'm familiar with the recent popularity of PIC micros, there's a lot to be said for sticking with an architecture you know. In my case, that means Motorola microcontrollers—in particular, the 'C8.

Its most attractive features are its 304 bytes of RAM and about 7 KB of EPROM. They enable modest amounts of data to be collected without external memory devices.

More importantly, I can download undebugged code into RAM. It's more efficient than burning EPROMs for each iteration of the development cycle. I supplement the relatively small RAM program area with a monitor (in 'C8 EPROM) that includes target applications which can be called by the RAM program being debugged.

As well, development systems use many of the same routines to provide feedback, so I rely on the client-server concept. Since the information is input to and presented by the host PC, I partitioned the software to maximize the host's work and minimize the micro's,

Photo 1--The complete development system, minus the wall-wart power supply, fits on a 4.5" x 6" proto-board.



The result is very lean monitor firmware, apart from general support routines. There's certainly room in the 'C8 EPROM for the monitor and a moderately complex target program to coexist.

The final stage is programming the device's EPROM directly from the host PC. This method replaces traditional methods described in the 'C8's technical manual.

There, the PROM programming circuit programs a 2764 EPROM in a conventional programmer and transfers it to the 'C8 programmer board, where a routine (in bootstrap ROM) copies it into the 'C8 EPROM. This somewhat cumbersome method calls for an EPROM programmer and ZIF sockets for both the micro and 2764.

CIRCUIT DESCRIPTION

The circuit in Figure 1 is quite similar to the Motorola design. Since the EPROM programming routine is handled by the code in the 'C8's bootstrap ROM, the wiring of the external EEPROM device to the 'C8 micro must follow Motorola's convention.

However, I replaced the 2764 with a 2864 EEPROM. Also, PC7 of the 'C8 now controls either the ● WR or ● OE signal of the EEPROM, depending on the mode. Switch S1A allows read access to the 2864 EEPROM during

verify and programming, and write access the rest of the time (Load mode).

Sections B and C of S1 control the application of V_{pp} programming voltage. They also shift the voltage applied to the ● IRQ pin from V_{CC} to the 9 V necessary to place it into Bootstrap mode.

For some reason (probably EPROM program pulse timing), Motorola specifies a 2.0-MHz clock for the programmer circuit. I'm using the circuit for real-time emulation, so I want the normal 4.0-MHz clock available as well.

I used a 4-MHz crystal in an oscillator composed of three sections of U1 (a 74LS04) and added U2 (a 4013 divide-by-2 circuit). Switch S1D selects which clock signal is fed to the 'C8's OSC1 pin. You can also use two oscillator modules—a 2-MHz and a 4-MHz one.

The link to the host PC is handled by a MAX232 single-chip RS-232 transceiver/charge pump, eliminating the need for a separate negative power supply to handle the RS-232's negative-signal excursions.

A ubiquitous 12-V AC wall-wart adapter supplies raw AC power. Since the total current draw is only 125 mA, the actual AC voltage from a nominal 12-V adapter is somewhat higher (usually 14 V).

Using a full-wave bridge rectifier provides -18 V DC. The 14.75 V needed for programming is provided by an

LM317 three-terminal adjustable regulator.

The actual V_{pp} voltage is critical, so adjust R20 to provide exactly 15.5 V at the LM317's output terminal *before* a 'C8 device is placed in the ZIF socket.

The drop-out voltage of an LM317T with normal programming current at room temperature is 1.6 V. So, your adapter must put out enough voltage to obtain at least 17.1 V DC (14.75 V_{pp} + 0.75 V D4 diode drop + 1.6 V LM317 dropout) across C7.

Hint: Since there's so little room in this circuit, make sure that the LM317 actually regulates by adjusting R20 through its range while ensuring that the output voltage changes, and leave it at 15.5 V when finished.

A 7805 regulator provides 5 V. The 9 V for the Bootstrap mode is tapped off the 12-V supply provided by a 1N4742 zener diode.

Rounding out the circuit is the 40-pin target header connector. It connects via a 40-conductor ribbon cable and DIP socket to the target board.

I connected PORT A, B, C, most of D, and ● IRQ to this header. The ground pin is connected, but V_{cc} , clock, and *RESET signals are not.

Since ports A, B, and C are also used to interface to the EEPROM, the EEPROM must be removed from its socket. Photo 1 shows the development board with the mode switch connected up through a multiconductor cable.

HOST SOFTWARE

Writing the firmware took the most time. The 'C8 monitor was the hardest to debug, but the host-PC software is far and away the most complicated, doing the bulk of the work.

The success of many good DOS-based programs and development boards stems from the fact that they often contain a debugger, communication utility, and assembler (or compiler) in one integrated package.

I wrote a Windows-based program that lets me use a shareware assembler while still enabling me to switch rapidly between the assembler and development-board software.

During debugging, you're constantly shifting among displays of RAM, vari-

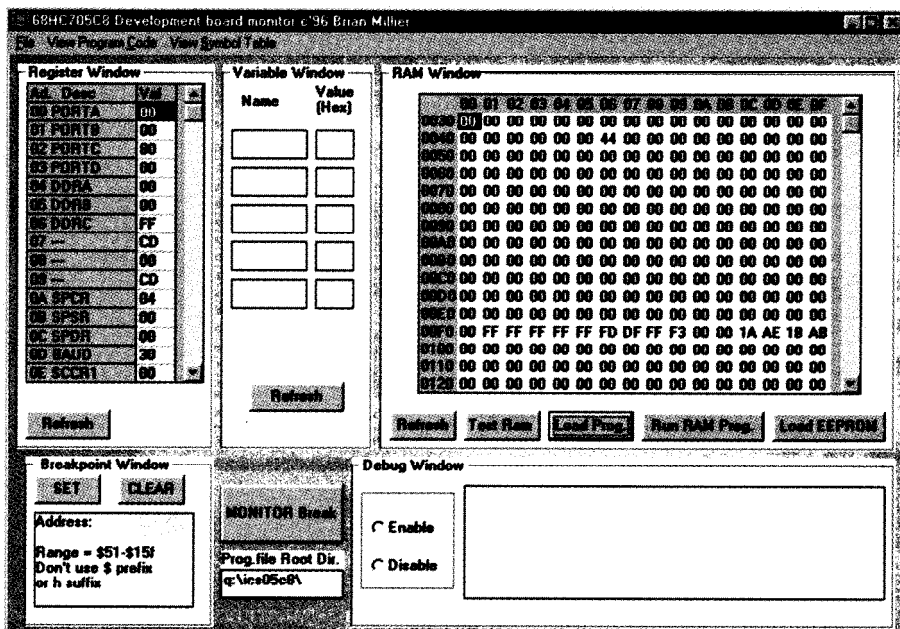


Photo 2—In this screen display of the Windows program running on the host PC, there are independent windows for RAM, variables, and registers.

ables, registers, symbol tables, and source code. Windows programs are ideally suited for that.

Although other programming languages can be used, given Windows, I went with Microsoft Visual Basic. It has many features that work well in data collection and manipulation.

Photo 2 shows the main window of the host program, C8MON 1. The following sections describe each window.

RAM WINDOW

The RAM list box displays the 'C8's RAM contents. To change a RAM location, click on the desired cell, enter the two hex digits, and press the space bar. The new data is immediately sent to the 'C8.

The **Test RAM** button fills all RAM with FFh and then 00h and verifies each pattern. This function is also useful for initializing all variables to zero.

The only area it leaves untouched is the area from F0h to FFh. The stack is fixed at FFh after reset (or **RS P**), and I leave 16 bytes untouched by the RAM test routine to accommodate a modest stack area.

The **Load Prog.** button loads a program into RAM. At this stage, you select the desired .S 19 format file.

I follow Motorola's RAM bootstrap load convention, which specifies that RAM programs all start at 51h, leaving the 30h-50h area free. I placed the 'C8 monitor's variables there.

The RAM area is divided into three sections, starting at 30h. The sections encompassing 30h-4Fh and 100h-015Fh are optional RAM areas. At

reset, they are mapped by default to internal EPROM in the 'C8.

If the user writes a 1 to bit 7 of the OPTION register (at 1FDFh), RAM is mapped into the 30h-4Fh region. Similarly, writing a 1 to bit 6 maps RAM into 100h-15Fh.

At this point, I must bring up a beef I have with Motorola on memory mapping. Clearly, 304 bytes of contiguous RAM are available in the maximum RAM configuration.

I use the term "contiguous" rather loosely, since Motorola fixed its stack area in the middle of this block (at FFh). Since the 50h-FFh block is the only area that is always RAM, it makes sense that the stack resides there somewhere.

However, you cannot load a larger RAM-based program, since it encroaches on the stack area. To get around this problem, you can tailor the source code to leave the stack region alone.

First, watch the listing file produced by the assembler as the program is written. When the file approaches F0h, place an **ORG \$10 0** directive in the code to restart assembly at that location, thereby jumping over the stack region.

The RAM load routine (in the C8's monitor EPROM) skips over any references to addresses in the F0h-FFh range for stack overwrite protection.

Once the RAM is loaded, **Run RAM program** starts program execution at 51h and enables the debug window. Now, you can interact directly with the running 'C8 program through the SCI port.

Routine	Address	Description
Add16	\$1B91	16-bit add
Binasc	\$1833	Convert binary word into 4 ASCII digits
Binhex	\$1A89	Convert byte in A into 2 hex digits
Byte	\$1A64	Convert two ASCII hex digits to 8-bit binary value
Delay	\$1C3C	Value in A determines delay in milliseconds
Getvalue	\$1D0C	Read up to 3 keypad entries and convert to binary
Keyscan	\$1C4A	Scan I&key pad connected to Ports A, B
LCD_init	\$1B9C	Initialize LCD. LCD is connected for 4-bit transfers
LCD_clr	\$1BD9	Clear LCD screen and home cursor
LCD-write	\$1BE2	Write char in A to LCD at current cursor location
LCD_movcur	\$1BE9	Move cursor to position in A
LCD_writec	\$1BF2	Write BCD nibble as ASCII digit
RTCstart	\$1CC0	Start up real-time clock (uses Timer 0)
Sub16	\$1B86	16-bit subtract
SClin	\$1AAC	Interrupt-driven SCI input with 1-character buffer
SClout	\$1AB5	Output value in A to SCI output
Waitedge	\$1B06	Measure time interval between two Input Captures

Table 1—These useful routines as well as their entry points are available in the C8MON1 firmware

ALL ELECTRONICS CORPORATION

THERMOELECTRIC COOLER PELTIER JUNCTIONS I



Current applied to the device will produce heat on one side and cold on the other side, up to 68° C difference between the two sides. Modules can be mounted in parallel to increase the heat transfer effect or can be stacked to achieve high differential temperatures. 127 thermocouples per device. Operates on 3-12 Vdc. Requires a heatsink to prevent overheating.

1.18" (30 mm) square X 0.15" (3.6 mm) thick.

\$17⁰⁰ each CAT# PJT-1
5 for \$75.00

1.57" (40 mm) square X 0.15" (3.8 mm) thick.

\$26⁰⁰ each CAT# PJT-2
5 for \$110.00

Quantity Pricing Available!

5 amp Solid State Relay

C.P. CLARE / Theta J # JTA2405-3

Compact, TTL compatible, optically isolated solid state relay for loads up to 5 amps @ 240 vac.

0.8" x 0.82" x 0.56" high epoxy block with a 1.4" long metal mounting flange. 1.19" mounting centers. 0.062" dia. x 0.175" high pins. Pins can be pc mounted wrapped and soldered UL and CSA listed. Input: 4-8 Vdc Load: 5 amos @ 240 Vac



\$6⁰⁰ each CAT# SSRLY-2405
10 for \$55.00

470 UF, 450 VOLT SNAP-IN CAPACITOR

Nichicon LGQ2W471MHSC
1.375" diameter x 2" high. 0.4" lead spacing.



\$4⁵⁰ each CAT# EC-4745
10 for \$40.00

ORDER TOLL FREE

1-800-826-5432

CHARGE ORDERS to Visa, Mastercard, American Express or Discover

TERMS: NO MINIMUM ORDER. Shipping and handling for the 48 continental U.S.A. \$5.00 per order. All others including AK, HI, PR or Canada must pay full shipping. All orders delivered in CALIFORNIA must include local state sales tax. Quantities Limited. NO COD. Prices subject to change without notice.

CALL, WRITE FAX or E-MAIL for our FREE

96 Page CATALOG

Outside the U.S.A. send \$3.00 postage.

MAIL ORDERS TO: ALL ELECTRONICS CORPORATION P.O. Box 567

Van Nuys, CA 91408 FAX (818)781-2653

e-mail allcorp@allcorp.com

internet - http://www.allcorp.com

There are two methods for ending program execution and returning to monitor function. An SW I instruction may be placed after the final instruction that is executed in the program code.

Alternately, a breakpoint can be set at that location, using the breakpoint window. In either case, a breakpoint message is displayed in the debug window when this instruction is reached. The debug window remains active until you select `Display`.

`Refresh` updates the RAM list box with the current values in the 'C8's RAM. Since all data communication between the 'C8 and the PC host takes place at a modest 9600 bps, it makes sense to minimize traffic. So, rather than constantly updating the RAM and register list boxes, you can refresh the display as necessary.

The Load EEPROM button, while part of the RAM window, is used in programming the 'C8's EPROM and is described later.

VARIABLE WINDOW

It's convenient to be able to view variables by name after a program terminates. To select them, choose `View Symbol Table` and pick the .LST file corresponding to the program you loaded into RAM. (For this feature to work with my code, use P&E's IASM05 assembler, available on Motorola's freeware site.)

To add a variable to this window, select it from the list box and then click on the desired variable-name box. Photo 3 shows the symbol window after a few variables are chosen.

REGISTER WINDOW

This window provides direct access to the 'C8's hardware registers, which is handy for setting Data Direction registers as well as reading and setting ports.

To access a particular register, click on the desired `Value` cell and enter two hex digits followed by the space bar. That value is immediately sent to the

'C8, and its resulting value is read back and displayed.

Note that if you write values to registers with read-only bits or write to ports that are defined as inputs, the displayed value may differ from the value you originally entered.

Access to SCI registers is blocked from this window to prevent a user from inadvertently redefining any aspect of the SCI port, potentially destroying the 'C8-to-host data link and crashing the monitor. However, repeatedly pressing `Refresh` while viewing the timer registers displays their constantly changing values.

BREAKPOINT WINDOW

This puny window lets you set or clear a breakpoint at a chosen RAM location. The opcode there is replaced by an SW I instruction (but saved and reinstated if a clear is performed).

Since RAM-based programs are simple due to their limited size, a more sophisticated breakpoint facility

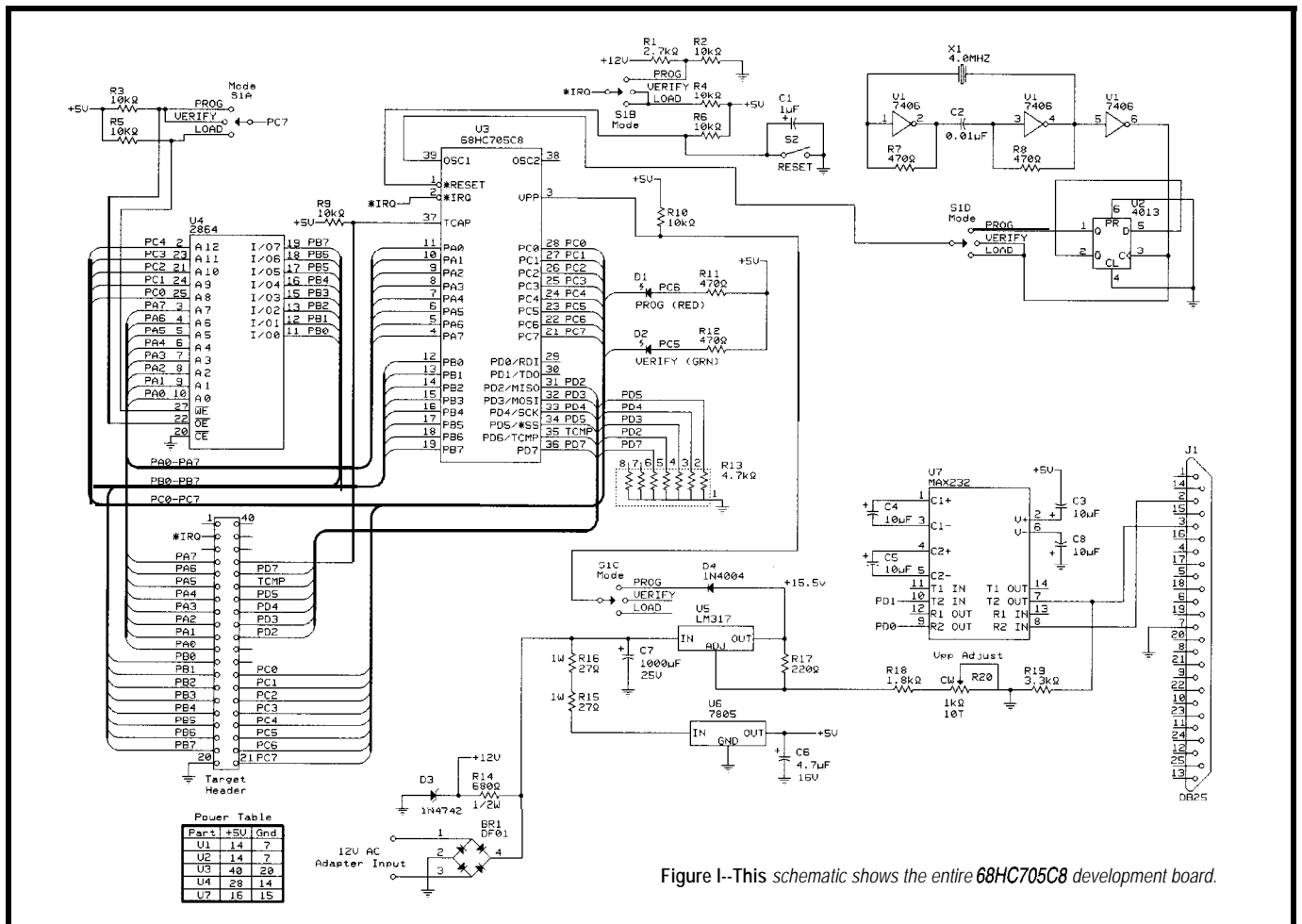


Figure 1—This schematic shows the entire 68HC705C8 development board.

isn't necessary. I considered providing more support but was stymied when I realized the 'C8 has no stack-manipulation instructions!

Thus, it seems impossible to determine where a program is when an SW I (or any interrupt) occurs. If you have a clever solution, I'd be pleased to hear from you!

DEBUG WINDOW

When a RAM-based program is running, it can be useful to send or receive data to the 'C8 SCI port in an unrestricted, free-form format. When enabled, the debug window does exactly that.

Say, for example, you wrote a program to echo the SCI port. When you run a RAM program in this system, the debug window is enabled as soon as the 'C8 program starts executing.

If you click inside the large data window in the debug section, any characters you type at the host PC are sent to the 'C8 SCI port. Characters received from the 'C8 are also displayed.

To restore normal monitor operation, the 'C8 and the host PC must be returned to the monitor function and any program executing on the 'C8 must finish.

If your program doesn't have an SW I after the last instruction it executes or if a breakpoint is not encountered, press MONITOR break. A short sign-on message appears in the debug data area, followed by a prompt. D i s a b l e exits Debug mode and returns to normal monitor operation.

This procedure should also be followed if normal monitor operation becomes disrupted, as indicated by a time-out message on the host PC's screen.

OTHER FEATURES

Under the file menu is an entry to select the default directory where the 'C8 program files are located. The default directory entry is stored in the C8MON1.INI file.

View Program Code lets you open a window containing source code of the program you are working with (or

any other 'C8 source code, for that matter). Text in this window is limited to 64 KB, which is a function of the VB control.

You can choose either .ASM or .LST files in the file dialog box. List files are more useful since they reference actual memory locations, but they're much larger. Size is no problem for RAM-based programs but larger programs meant for EPROM may be too big in .LST format.

SYSTEM OPERATION

To implement this development system, first download the code. The firmware for the 'C8 is C8MON1.S19, and its source code is C8MON1.ASM. The host PC software including the Visual Basic run-time package and necessary VBX files are in C8MON1.ZIP.

One advantage of my system over Motorola's programming circuit is its ability to download code from the PC directly, eliminating the need to burn 2764 EPROMs.

However, to program a 'C8 with the C8MON1 firmware, you need to get that

AVT inc.

Advanced Vehicle Technologies, Inc.

Products to support the design and testing of vehicle network components

AVT's Services...

- Automotive Network Engineering: J1850 VPW, PWM, & ISO-9141
- Hardware Design: Digital & Analog
- Embedded Software/Firmware Development
- PC Based Software Design & Development
- Custom Software & Hardware Development, Assembly, and Test

AVT's Products...

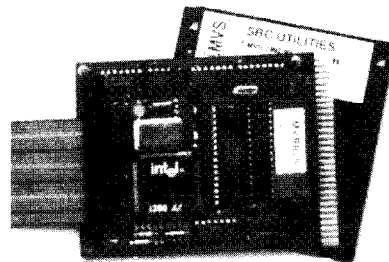
- AVT-716: Triple Interface J1850 (VPW & PWM) & ISO-9141 (RS-232/422)
- AVT-715: Dual J1850 Interface VPW & PWM (RS-232/422)
- AVT-921: Dual J1850 Interface VPW & PWM (ISA Bus)
- AVT-1850: J1850 VPW Development System

"Vehicle network expertise, products, and resources"

1509 Manor View Road • Davidsonville, MD 21035
(410) 798-4038 voice, (410) 798-4308 fax

e-mail: avt-inc@ari.net • home page: <http://www2.ari.net/avt-inc/>

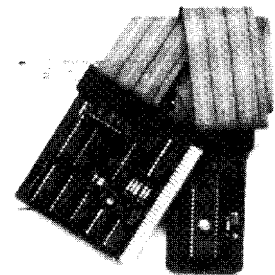
386 SBC \$83



OEM (1K) PRICE INCLUDES:
 - 5 SER (8250 USART)
 - 3 PAR (32 BITS MAX)
 - 32K RAM, EXP 64M
 - STANDARD PC BUS
 - LCD, KBD PORT
 • BATT. BACK. RTC
 • IRQ0-15 (8259 X2)
 • 9237 DMA 8253 TMR
 - BUILT-IN LED DISP.
 - UP TO 8 MEG ROM
 - CMOS NVRAM
 USE TURBO C, BASIC, MASM RUNS DOS AND WINDOWS
 EVAL KIT \$295

\$95 SINGLE PRICE PIECE UNIVERSAL PROGRAMMER

-DOES 8 MEG EPROMS
 - CMOS, EE, FLASH, NVRAM
 - EASIER TO USE THAN MOST
 • POWERFUL SCRIPT ABILITY
 - MICROCONT. ADAPTERS
 - PLCC, MINI-DIP ADAPTERS
 - SUPER FAST ALGORITHMS



OTHER PRODUCTS:
 8088 SINGLE BOARD COMPUTER OEM \$27 ... *95
 PC FLASH/ROM DISKS 128K-16M) 21 75
 16 BIT 16 CHAN ADC-DAC CARD 55 195
 WATCHDOG (REBOOTS PC ON HANGUP) 27 95

EVAL KITS INCLUDE MANUAL BRACKET AND SOFTWARE.
 5 YR LIMITED WARRANTY
 FREE SHIPPING
 HRS: MON-FRI 10AM-6PM EST

MVS MVS BOX 850
 MERRIMACK, NH
 (508) 792 9507

code into either a 2764 or 2864 EEPROM which will end up in the circuit. Be sure to remove power from the circuit before inserting or removing the 'C8 in its ZIF socket.

With a blank 'C8 and (E)EPROM device (with C8MON1 loaded) on the development board, place the processor in the Reset position using S2, select Program mode using S1, and release S2 from Reset.

The red LED lights for less than a minute during programming. If all goes well, the green LED indicates that verification completed successfully.

Return S1 to the Load position. Load is the normal operating mode for the monitor, apart from EEPROM verification and 'C8 programming.

If you connect the unit to the host PC and start C8MON1 (from Windows), the development system should be functional. A quick click on RAM Test or RAM Refresh should display results.

The first line of C8MON1.INI is the comm port number that the development system is hooked up to (the default is COM1). The second line is the path to your 'C8 assembly-language programs (the default is c:\i c s 0 5 c 8\).

From this point on, programming other 'C8s is easier. Start with a 'C8 containing C8MON1 in the ZIF socket. Run the monitor program and select Load EEPROM and an .S19 file. The

file takes some time to load since it takes 10 ms per byte to write the 2864 EEPROM.

After loading, the program prompts you to switch S1 to Verify mode. Make sure to return S1 to the Load position after verification is complete.

Place S2 in the Reset position, and power down the development board. Replace the 'C8 device containing the monitor with the blank device to be programmed.

Before restoring power, place S1 in Program mode and power up. Release S2 from Reset, and the blank 'C8 device will be programmed. When the green LED comes on, verification is complete and the unit can be powered down.

Restore S1 to the Load position to prepare for the next time the unit is used. I've forgotten this important last step and reprogrammed my monitor 'C8 when I powered the board up the next time in Program mode!

APPLICATION CODE

Developing assembly code for a small microcontroller is never easy. Even if your program logic is flawless, timing considerations or interrupt complications often throw a monkey wrench into the process.

Here's my suggested plan of attack. After defining the hardware and soft-

ware tasks, use the C8MON1 firmware utility routines to design some of the basic building blocks of your program.

If your program uses the SCI port, use those routines. The firmware includes support for common LCDs, keypad scanning, and a real-time clock.

Table 1 lists some of the monitor utilities. Insert lines at the top of your code defining entry points for the routines you use (e.g., SCIOUT equ \$1AB5).

Write short RAM-based programs to exercise all I/O functions. For example, if you need to measure a pulse width using Input Capture, call that routine in the monitor, convert it to ASCII via BinAsc, and send it out the SCI port or LCD.

Test code fragments with short RAM programs. Use either the SCI or LCD to output results, or leave the results in RAM and examine those variables from the monitor after the program executes.

The RAM can only hold small programs, so at some point, you need to integrate the routines into an EPROM program. As well, be sure to integrate the monitor functions you used into your own assembly code.

Since there is not a lot of RAM available when the monitor runs RAM-based programs, many monitor routines use temporary variables. While the monitor performs normal duties, there's no conflict with this sort of variable sharing.

Once you migrate your program to an EPROM-based one, check the details (e.g., define the vector table at 1FF4h, enable or disable interrupts as needed, set the Option register for proper memory mapping, etc.).

If you're using an "A" version of the 'C8, some changes are necessary. In particular, program MOR1 and MOR2 to 0s to ensure, among other things, that the nonprogrammable COP is disabled. Check the technical manuals and app notes for details.

ODDS AND ENDS

If your application software isn't too complex and you've used a lot of the monitor utility routines, you may wish to include C8MON1 as part of your target code. It takes up less than 2 KB of the 'C8's available 7.7-KB EPROM.

Name	Location	Erase
CGRAMTABLE	1AF3	Erase
CHXVU	1917	Erase
CHXVUA	1903	Erase
		Erase
		Erase

To add a variable to the Variable Window:
Select an item from the list, end click on the desired variable name box

BH1	19D5
BH3	19D7
BH4	19E7
BH5	19E9
BINHEX	19C9
BRATE	000D
CB1	194C
CGRAMTABLE	1AF3
CGRLOOP	1AE3
CH1TARG	0030
CH2TARG	0031
CH3TARG	0032
CH4TARG	0033
CH5TARG	0034
CH6TARG	0035
CH7TARG	0036
CH8TARG	0037
CHVTARG	0038
CHPTB	003D
CHXVU	1917
CHXVUB	190D
CLEARBUF	1949
COPCR	001E
COPRST	001D
D1	1B77
D2	1B78
DDRA	0004
DDRB	0005
DDRC	0006

Photo 3—Values for the variable window are selected from the symbol table, which is derived from information present in the assembler's listing file.

After Reset, C8MON1 firmware makes PORT C an output. It then initializes the SCI to 9600 bps, enables the received-character interrupt (for interrupt-driven SCI input), and sets the Option register to enable both optional blocks of RAM.

The C8MON1 firmware then turns on both LEDs and raises PC7 to disable the EEPROM from either read or write access. It also enables interrupts.

At this point, it checks the state of port line PD2. This line, as well as PD3-5, is pulled low by the R13 resistor pack. These lines must be low when programming the 'C8 EPROM since they define the Bootstrap mode.

However, once the required programming is done, if a jumper is placed between PD2 and V_{cc}, the C8MON 1 firmware detects this at startup, jumps to EPROM location 0160h, and runs the user code loaded there.

To get your code there, ORG it at 160h, paste it into C8MON1.ASM, and assemble the file.

As for other routines, a real-time clock with 0.1-s resolution is implemented using the Timer output compare function/interrupt and is started via RTCStart.

A 16-bit pulse counter routine is implemented using the *IRQ line and associated interrupt. Details are in the monitor listing, but don't try using the ● IRQ line for other purposes.

The SCI input routine is interrupt driven, so keep that in mind if you use the SCI without using my routines.

WRAP UP

I used this project in developing a mini-network of pressure-monitoring data stations for physical-chemistry lab experiments.

The 'C8 is ideal because it contains enough RAM to hold the collected data and enough ports to handle the I/O.

After each experiment, the student sends data to the host PC for processing, recording, and printing. The data stations use the 'C8 SCI port with a modified RS-232 driver circuit, so multiple stations can be connected on one RS-232 line.

This setup lets you use the PC's built-in RS-232 port, eliminating any

need for a dedicated RS-422 or -485 board.

Even if you're not particularly interested in the Motorola 'C8, I hope you can benefit from some of the concepts in this design—especially the client-server technique used in the monitor.

Also, there are many interesting tools available in Visual Basic which serve data acquisition and manipulation applications well.

You'll likely find fertile ground for your own article as well. □

Brian Millier has worked as an instrumentation engineer for the last 15 years in the chemistry department of Dalhousie University, Halifax, NS, Canada. He also operates Computer Interface Consultants. you may reach him at brian.millier@dal.ca.

SOFTWARE

The C8MON1 firmware and Visual Basic host-program executable code are available through the Circuit Cellar Web site.

SOURCES

MC68HC705C8

Motorola
MCU Information Line
P.O. Box 13026
Austin, TX 7871 1-3026
(512) 328-2268
Fax: (512) 891-4465

MC68HC705C8 Technical Data manuals, AN1226

Motorola Literature Distribution
P.O. Box 20912
Phoenix, AZ 85036
design-net.com/CSIC/TECHDATA/DATABOOK/datalist.htm

MAX232

Maxim Integrated Products
120 San Gabriel Dr.
Sunnyvale, CA 94086
(408) 737-7600
Fax: (408) 737-7194

IRS

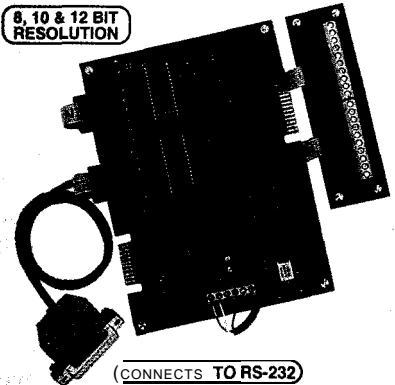
407 Very Useful
408 Moderately Useful
409 Not Useful



AR-16 RELAY INTERFACE (16 channel) \$ 89.95
Two 8 channel (TTL level) outputs are provided for connection to relay cards or other devices (expandable to 128 relays using EX-16 expansion cards). A variety of relays cards and relays are stocked. Call for more info.
AR-2 RELAY INTERFACE (2 relays, 10 amp).... \$ 44.95
AR-6 REED RELAY CARD (6 relays, 10 VA) \$ 49.95
RH-8 RELAY CARD (10 amp SPDT, 277 VAC).... \$ 69.95

ANALOG TO DIGITAL

8, 10 & 12 BIT RESOLUTION



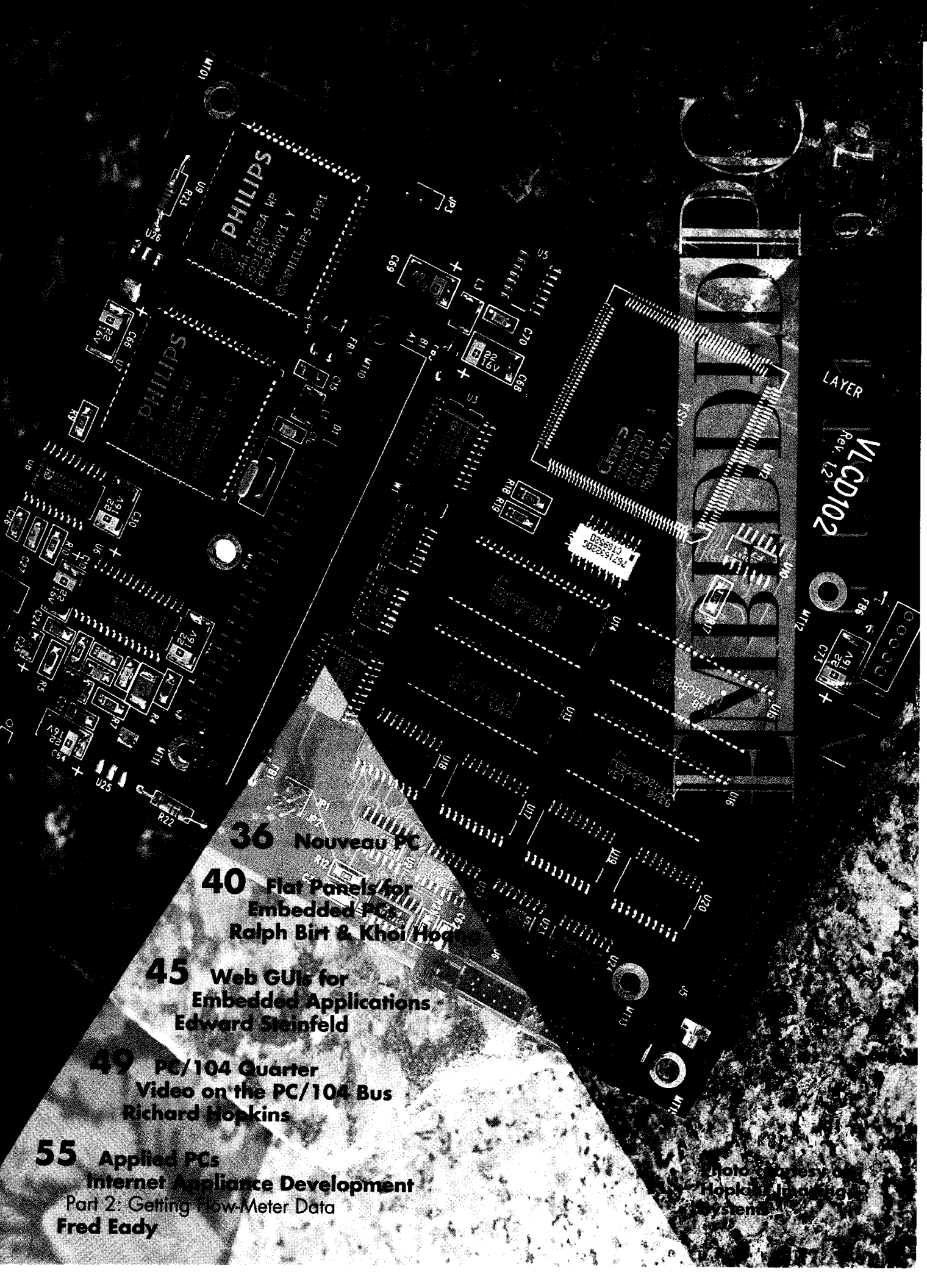
ADC-16 A/D CONVERTER (16 channel/8 bit) ..\$ 99.95
ADC-8G A/D CONVERTER (8 channel/10 bit) ..\$124.90
Input voltage, amperage, pressure, energy usage, light, joysticks and a wide variety of other types of analog signals. RS-422/RS-485 available (lengths to 4,000'). Call for info on other A/D configurations and 12 bit converters (terminal block and cable sold separately). Includes Data Acquisition software for Windows 95 or 3.1
ADC-8E TEMPERATURE INTERFACE (6 ch.) ..\$ 139.96
Includes term. block & 8 temp. sensors (-40' to 146' F).
STA-8 DIGITAL INTERFACE (6 channel) \$ 99.95
Input on/off status of relays, switches, HVAC equipment, security devices, keypads, and other devices.
PS-4 PORT SELECTOR (4 channels **RS-422**)... \$ 79.95
Converts an RS-232 port into 4 selectable RS-422 ports. CD-422 (RS-232 to RS-422 converter) \$ 39.95
***EXPANDABLE**...expand your interface to control and monitor up to 512 relays, up to 576 digital inputs, up to 128 analog inputs or up to 128 temperature inputs using the PS-4, EX-16, ST-32 & AD-16 expansion cards.

• **FULL TECHNICAL SUPPORT**...provided over the telephone by our staff. Technical reference 8 disk including test software & programming examples in QuickBasic, GW Basic, Visual Basic, Visual C++, Turbo C, Assembly and others are provided.
• **HIGH RELIABILITY**...engineered for continuous 24 hour industrial applications with 10 years of proven performance in the energy management field.
• **CONNECTS TO RS-232, RS-422 or RS-485**...use with IBM and compatibles, Mac and most computers. All standard baud rates and protocols (50 to 19,200 baud).
FREE INFORMATION PACKET...use our 800 number, Fax or E-mail to order, or visit our Internet on-line catalog.
URL: <http://www.eecel.com>
Technical Support (614) 464-4470

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

Internet E-mail: eecl@ibm.net
International & Domestic FAX: (614) 464-8656
Use for information, technical support & orders.

ELECTRONIC ENERGY CONTROL, INC.
360 South Fifth Street, Suite 604
Columbus, Ohio 43215-5491



LAYER
V1 CD102
Rev. 1.2

36 Nouveau PC

40 Flat Panels for Embedded PCs
Ralph Birt & Khai Hoang

45 Web GUIs for Embedded Applications
Edward Steinfeld

49 PC/104 Quarter Video on the PC/104 Bus
Richard Hopkins

55 Applied PCs
Internet Appliance Development
Part 2: Getting Flow-Meter Data
Fred Eady

Photo Courtesy of
Hopkins Technology
Systems

DISK-EMULATOR CHIP

The PROMDisk-Chip disk-emulator chip is designed to plug directly into any standard 32-pin EPROM socket in the BIOS extension address space of a PC-compatible computer. The Chip is bootable as a fixed disk ranging from 2 to 8 MB by using nonvolatile NOR flash memory. It includes an internal BIOS extension ROM that

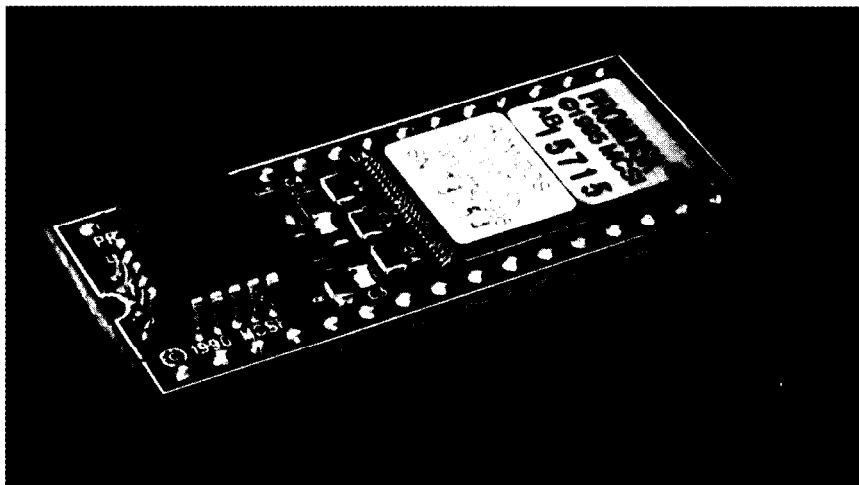
houses the Datalight VBF integrated flash file system and ROM-DOS 6.22. It is fully DOS and Windows compatible, enabling the user to copy and erase files using standard DOS commands.

PROMDisk-Chip replaces mechanical disk drives in systems operating in harsh environments or where temperature, shock, vibration, or reliability are concerns. In embedded or dedicated applications, the Chip offers substantial benefits in overall system cost, performance, and reliability. Since it runs at bus speeds and is not encumbered by mechanical latency, the average read/write throughput is dramatically increased over that of a typical hard disk drive.

The 4-MB PROMDisk-Chip disk-emulator chip sells for \$150. It comes complete with the on-board

Datalight VBF flash file system, Datalight ROM-DOSV.6.22, a user manual, and utility diskette.

Micro Computer Specialists, Inc.
2598-G Fortune Way • Vista, CA 92083
(760) 598-2177 • Fax: (760) 598-2450 **#510**



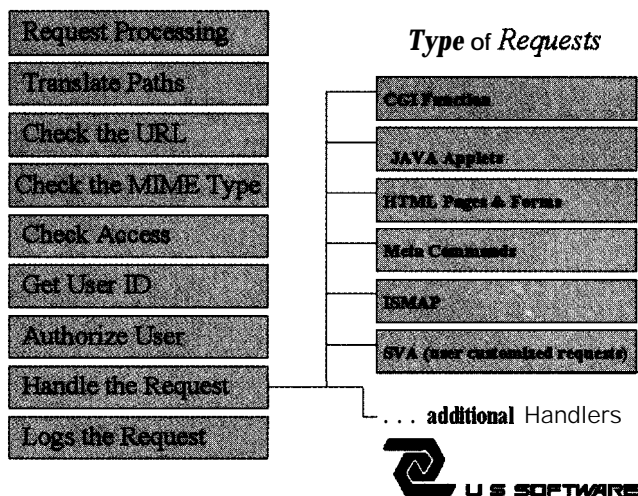
USNet INTERNET ACCESS PACKAGE

U.S. Software has announced the **Internet Access** Package (IAP) for embedded applications. It enhances USNet, the company's real-time embedded TCP/IP protocol stack, with Web-enhanced technology and may be used on LANs and WANs with an off-the-shelf Web browser.

USNet IAP is a standard Internet/Intranet protocol technology designed specifically for real-time embedded applications. It has a small footprint, high performance, and APIs designed for embedded applications. In typical Web-enabled applications running an embedded HTTP server on a 16-bit target (e.g., 'x86), the USNet protocol stack requires less than 25 KB for networking and 10 KB for the embedded Web server. USNet IAP includes dial-up capabilities, DNS resolver, E-mail protocols (SMTP, POP, and MIME), as well as an embedded HTTP server that supports CGI scripts, Java applets, Server Meta Controls, and ISMAP.

CGI scripts can easily be written by developers as C function calls to be passed between the embedded HTTP server and a standard Web browser. HTML forms and pages provide a de facto standard for developing projects and presenting embedded information in a style that includes graphics.

The IAP for USNet is priced at \$1000.

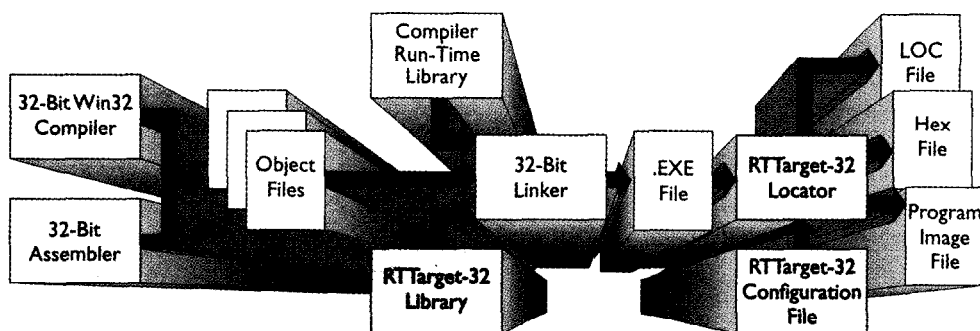


U.S. Software
142 15 NW Science Park Dr.
Portland, OR 97229
(503) 64 1-8446
Fax: (503) 644-24 13
www.ussw.com

#511

Nouveau PC

edited by Harv Weiner



A comprehensive run-time library is provided on the target. Part of the Win32-API (Application Program interface) is emulated, supporting software originally developed for DOS or Win32. The C/C++ run-time system is also supported, and additional functions for real-time applications are provided.

CROSS-DEVELOPMENT SYSTEM

A software licensing agreement between On Time and Paradigm Systems has produced a full-featured cross-development system. Under this agreement, On Time will integrate an enhanced version of Paradigm's Debug-32 product (based on Borland's Turbo Debugger-32) into its crossdevelopment system, **RTTarget-32**, V.2.0 and higher.

RTTarget-32 is designed for 32-bit embedded systems on the Intel '386 platform. It enables 32-bit programs developed for Windows 95 or NT to run without an operating system on an Intel '386, '486, or Pentium Pro. Popular C/C++ compilers are supported, and DOS and Windows systems can be used as a host.

RTTarget-32 provides boot code to start the target system from floppy, hard, EPROM, or flash disks or directly from ROM. The application program can then run directly from ROM or be loaded from disk. Optionally, the application can be downloaded using a serial link during the development phase.

Paradigm's Debug-32 lets users debug applications running on the target system with the familiar Turbo Debugger interface. Enhancements include awareness of data system structures (e.g., CPU descriptor tables, I/O ports, etc.) and the ability to be used with Borland C/C++ and Delphi, Microsoft C/C++, and Watcom C/C++. Other features include DLL support, data compression, and an improved structure for third-party libraries.

An RTTarget-32 2.0 developer's license is priced at \$1700. Complete source code is available for an extra \$1000. No run-time royalties are charged.

On Time
88 Christian Ave.
Setauket, NY 11733
(516) 689-6654
Fax: (516) 689-1172
www.on-time.com

#512

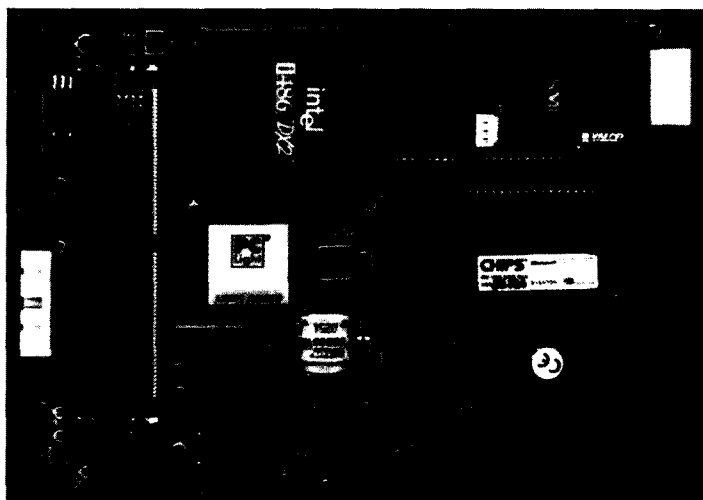
SINGLE-BOARD COMPUTER

Groupe Erim's Light Board 486 Features an Intel 486DX4 processor running at 100 MHz and supports up to 32 MB of 72-pin SIMM DRAM. Its 5.75" x 8" size holds the PC/104 16-bit bus, three RS-232 serial ports, one RS-232/485 port, a printer port, floppy controller, hard-disk interface, and mouse, keyboard, and I²C ports. Also included are a watchdog timer, 16 digital inputs, 8 digital outputs, VGA controller for CRT or LCD, Ethernet controller for 10Base-T, and flash memory up to 4 MB. For a disk application, DOS and software can be stored in onboard flash memory.

The Light Board 486 priced at \$700 is distributed in the U.S. by Gespac. A Pentium + Bus PCI version is also available.

Gespac
50 W. Hoover Ave. • Mesa, AZ 85210
(602) 962-5559 • Fax: (602) 962-5750
yves_bourdon@compuserve.com

#513



Nouveau PC

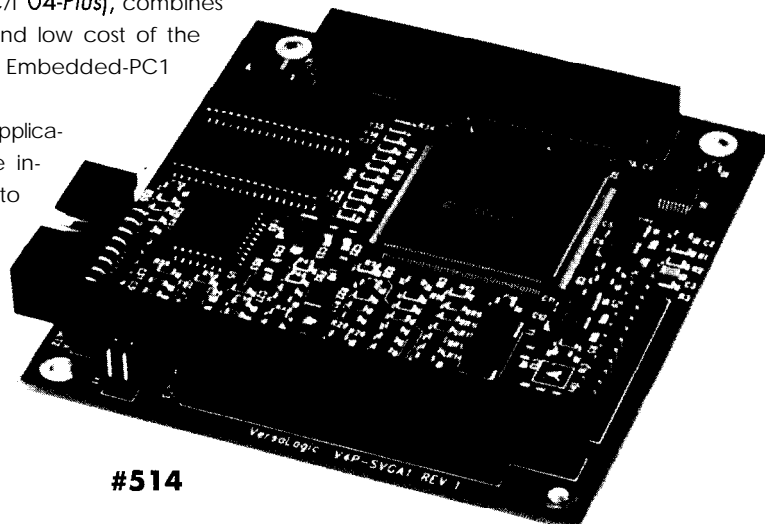
EMBEDDED-PCI SVGA MODULE

Versallogic has introduced an SVGA display module for the Embedded-PC and PC/I 04-Plus expansion interface. The EPM-SVGA-1 Display Module has been optimized for GUIs and operating systems like DOS, Windows 3.1, and Windows 95. Video drivers are included. As well, 1 MB of onboard video RAM allows color depth up to 16 million colors and screen resolutions up to 1280 x 1024 pixels.

The Embedded-PC architecture (also known as PC/I 04-Plus), combines the high-speed capabilities of PCI with the small size and low cost of the PC/I 04 bus. This new architecture allows both PC/I 04 and Embedded-PC1 modules to be stacked together in the same system.

The Embedded-PC1 architecture is useful for embedded applications requiring high-speed video capabilities. Its interface increases the data transfer speed from the PC/I 04's 5 MBps to 132 MBps.

The EPM-SVGA-1 sells for \$245 in OEM quantities.

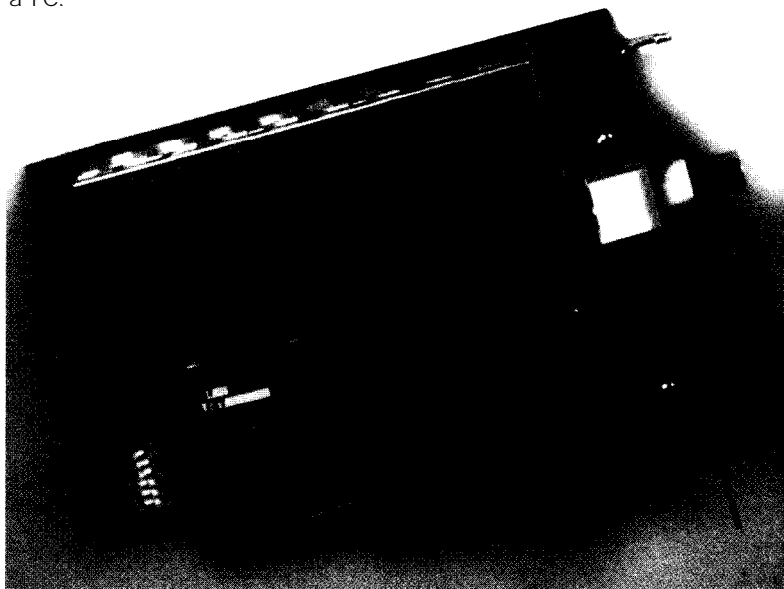


#514

Versallogic Corp.
3888 Stewart Rd.
Eugene, OR 97402
(541) 485-8575
Fax: (541) 485-5712
www.versallogic.com

FUNCTION GENERATOR

The Model ISA-104 Function Generator supplies sine-, triangle-, and square-wave outputs over a bandwidth of 1 Hz to 16 MHz. The instrument is a half-length ISA bus card that combines the features of a "box" function generator with the convenience of a PC.



Amplitude can be adjusted over a range of 100 mV to 20 Vp-p in 100-mV steps, with a maximum output of 20 Vp-p into a 50-ohm load. A variable DC offset of ± 10 V is available, and the duty cycle is variable over a range of 10-90%. Operating modes include free running, sweep, and crystal-controlled PLL-based modes. Square-wave rise time is less than 40 ns.

The driver and programmer's libraries included enable embedded control of the generator from user applications written in C, C++, or Visual Basic. Stand-alone software for DOS and a Windows GUI are also included.

The Model ISA-1 04 sells for \$395.

J-Works, Inc.
12328 Gladstone Ave., Ste. 4
Sylmar, CA 91342
(818) 361-0787
Fax: (818) 270-2413
jworksjm@gte.net

#515

Nouveau IPC

Ralph Birt
&
Khoi Hoang

Flat Panels for Embedded PCs

Today, users demand full windowing capability, even if it's not "needed" in their embedded system. Find out which flat-panel video controllers give users what they've come to expect from any computer-desktop or embedded.

Are you working on a compact or portable design? Are you wondering how you're going to put the GUI for a modern operating system on a small output display?

The cryptic PRNTR RDY message on a I-line x 1 O-character mode LCD is no longer socially acceptable. Today's users expect color, animated icons, point and click, drag and drop,....

You can argue about whether this "extra" stuff is needed on an embedded system, but it's how things are going and it's tough to swim against the tide. Try getting applications or support for a CP/M or DOS 2.1 machine. You'll know what we mean.

Fortunately, manufacturers of flat panels and flat-panel controllers have made it fairly easy to incorporate these devices into designs. Flat panels are bigger, brighter, cheaper, and easier to use than ever before. With some careful planning, you can wow users with the bells and whistles they've come to expect.

Our goal here is to provide some general guidelines and heuristics you can use for

your own flat-panel design. To that end, we discuss the advantages and disadvantages of a myriad of flat panels on the market.

We cover some of the different flat-panel video controllers produced by various manufacturers, design considerations regarding cabling and EMI, as well as the software tools available to facilitate interfacing to flat panels.

In general, flat-panel displays (FPDs) are electronic displays that are compact, light weight, and low power (compared to conventional CRTs).

Over the last several years, FPDs have gained the attention of the computer industry, especially in laptop and embedded applications. They're used in medical equipment, mobile computers, POS terminals, video lottery terminals, cockpit flight-display systems, and more.

Perhaps the hardest part in using this technology, especially for the embedded industry where only small quantities are manufactured, is the lack of standards (e.g., interface signals, connector, flat-panel

BIOS, etc.), which complicates things for designers.

FLAT-PANEL TYPES

Once you've decided to use a flat panel, you must first choose the type you want. Your choice depends on several factors, including but not limited to price, size, desired color, and power requirements.

So many different types of FPDs are available—liquid crystal displays (LCDs), light emitting diode displays (LEDDs), plasma display panels (PDPs), electroluminescent displays (ELs), vacuum fluorescent displays (VFDs), field emission displays (FEDs), and so on.

Since LCDs are, far and away, the most common type of FPD, we concentrate on them in this article.

LIQUID CRYSTAL DISPLAYS

LCDs are thin, lightweight, low-power, and low-voltage devices. Besides all those pluses, they're readily available from a variety of manufacturers.

LCDs are divided into three main groups—passive, active, and active-addressed. Passive matrix LCDs are the twisted nematic (TN) and super-twisted nematic (STN) displays you're probably most familiar with. Active matrix LCDs are the superior thin-film transistor types. Active-addressed LCDs combine these two technologies. Figure 1 illustrates the differences.

LCDs work by modulating light intensity from a CCFT (cold-cathode fluorescent tube) backlight. This modulation varies the control voltage across a liquid crystal cell.

The modulated light goes through a color (red, green, or blue) filter to produce the corresponding color portion (color dot) of a pixel. A pixel consists of adjacent red, green, and blue dots.

A typical 640 x 480 color LCD consists of an array of 640 x 480 x 3 color dots. In TFT LCDs, transistors control the liquid crystal voltage of these dots. LCDs now dominate the FPD market, and we don't expect their role to change in the near future.

TFT LCDs are the most important, widely used, and expensive member of the LCD family. Their image quality is as good as a CRT's, and they can display 24-bit (16 MB) color and generate no hazardous radiation. Also, they have a relatively fast response and low LCD supply voltage (5 V; 3.3 V is coming).

These compact, very low-power devices are widely available commercially. Recent improvements in TFT technology have enabled larger, higher resolution TFT LCDs to be manufactured at much lower prices.

The latest TFT LCDs on the market can produce 200-250 nits (cd/m^2). That's 3-4 times better than a conventional TFT LCD.

Moreover, their low reflectance, higher color saturation, wider viewing angle (70° horizontal), higher resolution (up to XGA at 150 dpi), and bigger size (up to 15") enable them to display color graphics images even in bright sunlight at a distance greater than arm's length.

Passive-matrix (mostly STN) LCDs once enjoyed the biggest market share due to their relatively low cost and acceptable quality. Now, they're being displaced from the top position by TFT LCDs.

Present technology can produce XGA 15" 150-nit passive-matrix LCDs with 70° horizontal viewing angle, over 240k colors with frame-rate modulation, and lower contrast voltage control (under 2 V, compared to about 30 V in a previous generation) at a lower price. However, their slow response and low contrast make them unsuitable for applications using video animations.

Monochrome LCDs are disappearing from the high-end-to-medium market because of the highly competitive price of STN LCDs.

EL DISPLAYS

The Et display consists of a solid-state glass panel, row-column control circuit, and high-voltage driver (100-200 V). As Figure 2 shows, the glass panel is doped with specific impurities (e.g., phosphor) to provide impurity states. Energizing the pixel of an intersecting row and column causes the light-emitting process to occur.

With their high brightness and contrast, wide viewing angle, low power consumption, fast response (< 1 ms), high resolution, long life, and large size, Et displays are suitable for a wide range of applications. You find them in the space shuttle, medical equipment, truck navigation terminals, cockpit display systems, and more.

Their major disadvantage is the lack of multicolor or fullcolor display capacity. These days, ELs can display either yellow, green, or red with a 16-level gray scale. Thanks to strong market demand, the development and availability of full-color ELs is anticipated in the near future.

FLAT-PANEL MANUFACTURERS

Sharp, with its 22-year history of LCD production, still gets the lion's share in the flat-panel market. Its LCD and Et flat panels fill up a long list, ranging from monochrome to large 15" XGA STN and 18-bit 14" XGA AM-TFT.

NEC, Toshiba, and FPD Company also manufacture high-quality AM-TFT FPDs. Or, to select an STN LCD, consider Sharp, Toshiba, Hitachi, Sanyo, Optrex, and Densitron.

Planar, with their Et ICEBrite family of 10.4", 8.1", and 6.4" VGA and quarter-VGA, is a good choice for medical equipment, vending machines, and car and cockpit navigation applications. These devices have exceptional brightness, wideviewing angle, long life, and endurance to extreme conditions (e.g., shock, vibration, temperature, and humidity).

FLAT-PANEL CONTROLLERS

While there are two main manufacturers of FPD controller-Chips and Technologies (C&T) and Cirrus Logic—newcomers such as ATI and S3 have arrived with some impressive products. Table 1 summarizes some of the important characteristics offered by these flat-panel controllers.

C&T supplies a OEM BIOS configuration utility called BMP (BIOS Modification Pro-

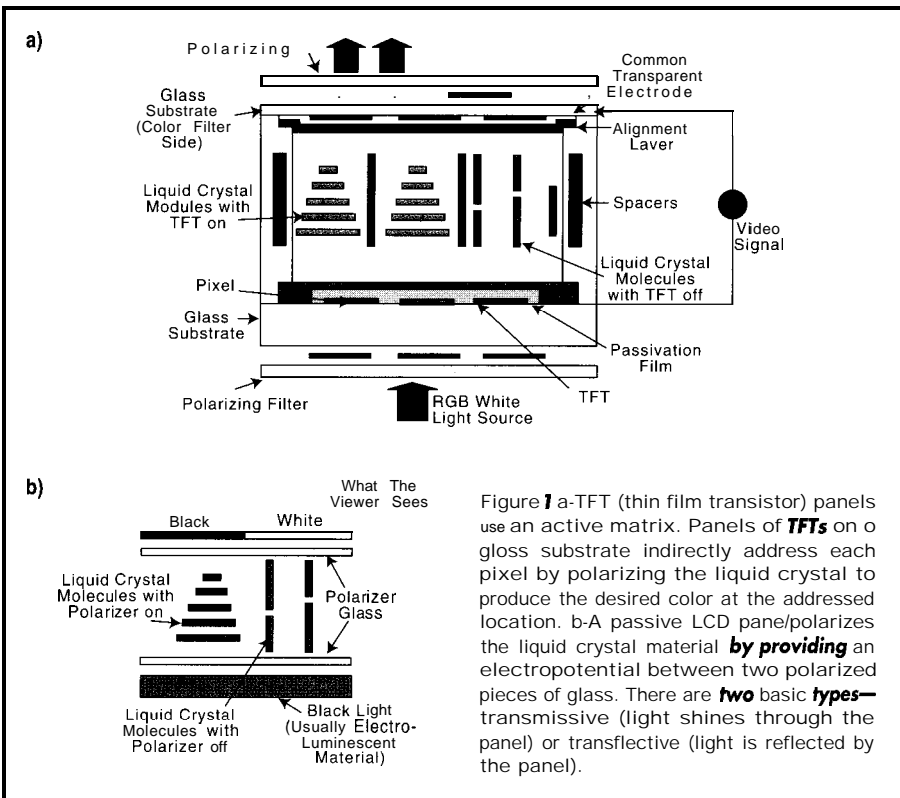


Figure 1 a-TFT (thin film transistor) panels use an active matrix. Panels of TFTs on a glass substrate indirectly address each pixel by polarizing the liquid crystal to produce the desired color at the addressed location. b-A passive LCD panel polarizes the liquid crystal material by providing an electropotential between two polarized pieces of glass. There are two basic types—transmissive (light shines through the panel) or transreflective (light is reflected by the panel).

gram) that's essential to interface with different flat panels. All C&T chips provide direct interface to virtually all existing monochrome and color STN, TFT, plasma, and Et flat panels.

The latest Cirrus Logic CRT/LCD controller families are the CL-GD754x and CL-GD755x.

The CL-GD7543 is a GUI-accelerated SVGA CRT/LCD controller with MVA (motion video acceleration) for MPEG playback and interface to an NTSC/PAL encoder. It's capable of running with either a 32-bit up to 50-MHz VL bus or a 32-bit PCI bus, and it supports color TFT and STN LCDs up to SVGA resolution. Its maximum 2 MB of video memory along with a high-performance GUI accelerator suit it for graphics-intensive applications.

The CL-GD7548 is the drop-in enhancement for CL-GD7543 with XGA capability and full MVA support for TFT and STN LCDs. The CL-GD7555 offers much higher video performance with its 64-bit GUI accelerator, while supporting up to XGA resolution. The latest CL-GD7556 is a low-power version of CL-GD7555.

Cirrus Logic supplies a OEM BIOS configuration utility called OEMSI to help interface with different flat panels without accessing the source code. Also, drivers are available for Windows 3.1 x, OS/2 Warp V.4.0 (Merlin), Warp 3.0 and 2.1 x, Windows 95, and Windows NT4.0 and 3.5x.

The Aurora64V+ Dual Display Accelerator is the first product from S3 to support flat panels. This chip has a built-in NTSC/PAL encoder to enable direct output to NTSC/PAL TV monitors.

DESIGN CONSIDERATIONS

The CRT/flat-panel controller is a high-speed, mixed-signal chip that requires special consideration for PCB layouts. A multilayer PCB with separate Vcc, analog V_{cc}, and GND plane is a must.

Table T-Here's a comparison of several of the more popular SVGA flat-panel control devices. Note how the features are becoming comparable to those available on CRT controllers.

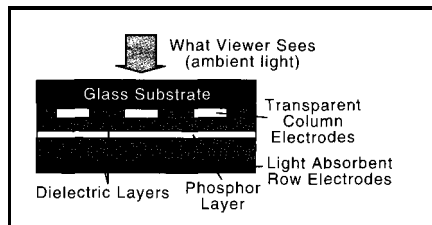


Figure 2—In an EL panel, potential is generated across the dielectric layers by a clocked voltage signal generated through the column and row electrodes, causing the desired pixel to light up because of the electropotential characteristics of the phosphor layer.

The clock-synthesizing power circuit must be well-isolated from digital noise. As well, all decoupling capacitors should be placed as close as possible to the video controller.

The traces from clock power pins must be routed directly through the pads of the decoupling capacitors, so don't leave any stub. And, don't route any high-frequency digital signals close to the analog sections. Keep in mind that isolated analog V_{cc}/GND islands are normally needed and recommended by chip manufacturers.

Because there's still no standard on how FPD signals are located on the interface connector or even on the type of connector used—making test cables for a new flat panel is complex and time-consuming.

This challenge is especially evident in embedded applications. Ordered quantities are usually small, and hundreds of different types and models of FPDs from many manufacturers have to be dealt with.

Doublecheck the cable before plugging in the FPD end. The old STN FPDs use very high positive-contrast voltage (15-30 V). Monochrome FPDs normally use very high negative-contrast voltage (from -15 to -30 V).

Always plug in the controller end of the cable first, measure voltage at every pin of the mating connector to the FPD's connector

at the other end, and verify against the wiring diagram. Connecting high voltage to a pin expecting low voltage can permanently damage the controller and the expensive flat panel.

In applications needing low EMI emissions or long distances between the controller and FPD, consider using LVDS from National Semiconductor or Panelink, a technology licensed by C&T.

The transmitter convert/multiplex display signals come from the video controller into differential signals with low voltages swing and send them over 4-6 cable pairs. They are then demultiplexed and restored at the flat-panel end. Newer flat panels come with built-in LVDS or Panelink receivers.

IN THE LONG TERM

Unfortunately, too often, you spend six months on a design only to find that the designed-in controller has been discontinued and is now difficult to obtain. The lifetime of any chipset can be extremely short.

It's always prudent to try to ensure the next generation of your controller will be a drop-in enhancement for the current controller.

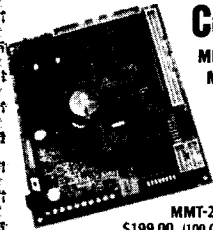
BIOS CONSIDERATIONS

Given the complicated characteristics of flat-panel interfacing, the BIOS should be as easily configurable as possible. Ideally, you want some utility software that can modify and dump the video-controller registers, enabling you to see the effects of any changes immediately on the display.

C&T's '5xx DEBUG and CHIPEXT utilities and Cirrus Logic's PCLRegs utility have proven helpful in developing BIOSs for new flat panels. Available drivers supporting different kinds of platforms are also an important factor.

Company	Chip	Bus support	Max Memory (MB)	GUI Accelerator	Power Sequencing	FP Interface	Package
C&T	65530	ISA	0.5	No	Yes	18 bit	PQFP
	65535	ISA/VL	0.5	No	Yes	18 bit	PQFP
	65540	ISA/VL	1.0	No	Yes	24 bit	PQFP
	65545	ISA/VL/PCI	1.0	No	Yes	24 bit	PQFP
	65548	VUPCI	1.0	Yes	Yes	24 bit	PQFP
	65550	PCI	2.0	Yes / 32 bit	Yes	24 bit	PQFP
	65554	PCI	2.0	Yes / 64 bit	Yes	24 bit	BGA
Cirrus Logic	CLGD7543	VUPCI	2.0	Yes	Yes	24 bit	PQFP
	CLGD7548	VUPCI	2.0	Yes	Yes	24 bit	PQFP
	CLGD7555	VUPCI	2.0	Yes / 64 bit	Yes	24 bit	PQFP
	CLGD7556	VUPCI	2.0	Yes / 64 bit	Yes	24 bit	PQFP
S3	Aurora64 +	PCI	4.0	Yes / 64 bit	Yes	24 bit	PBGA

80251 Embedded Controllers



MMT-188EB
MMT-Z180
MMT-HC11
MMT-196
MMT-31
MMT-251
MMT-PIC44
& OTHERS

MMT-251
\$199.00 (100 Quantity)

Midwest Micro-Tek is proud to offer its newest line of controllers based on the 8031/51/251 architecture. The 8031 comes in at a surprisingly low cost of \$89.00 (100 quantity). Custom work welcome.

MIDWEST MICRO-TEK
2308 East Sixth Street
Brookings, SD 57006
Phone 605.697.8521
Fax 605.692.5112
www.midwestmicro-tek.com

Guaranteed to be in your Future



#118

AMX The Real-Time Multitasking Kernel

680x0, 683xx PowerPC™ family
80386 protected mode i960® family
80x86/88 real mode R3000, LR33xxx

NEW • PowerPC support
• KwikLOOK™ AMX source code

- Compact, ROMable, fast interrupt response
- Preemptive, priority based task scheduler
- Mailbox, semaphore, resource, event, list, buffer and memory managers
- Configuration Builder utility
- Comprehensive documentation
- No royalties, source code included

For a sample of KwikLOOK and description of AMX.

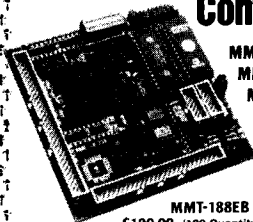
Phone: (604) 734-2796
Fax: (604) 734-8114
E-mail: amxsales@kadak.com
Web: http://www.kadak.com



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

#120

Low Cost Embedded Controllers



MMT-188EB
MMT-Z180
MMT-HC11
MMT-196
MMT-31
MMT-251
MMT-PIC44
& OTHERS

MMT-188EB
\$199.00 (100 Quantity)

If you're interested in getting the most out of your project, put the most into it. Call or Fax us for complete data sheets and CPU options. Custom work welcome.

MIDWEST MICRO-TEK
2308 East Sixth Street
Brookings, SD 57006
Phone 605.697.8521
Fax 605.692.5112
www.midwestmicro-tek.com

Guaranteed to be in your Future



#121

SELECTING AN FPD

Thin, lightweight, and low-power flat panels, with the help of small and powerful embedded computers, have created new opportunities for systems designers. It's now possible to develop highly portable and compact products with sophisticated and user-friendly GUIs—products we couldn't even imagine in the CRT era.

The most important step in designing new products is selecting correct components. At this stage, it's important to consider the environment the product is designed for and the kind of application it's for, as well as its cost, lifetime, and upgradability.

If the product is planned for areas where only low-to-medium ambient light is expected, LCDs are a good choice. Otherwise, ELs or new high-nit TFT LCDs are likely the best candidates. In extreme environments with high temperature, humidity, and vibration under direct sunlight, consider only ELs.

If the application is graphics animation with very high color-information content (e.g., in-flight VOD system, games, multimedia systems, medical imaging applications, etc.), TFT LCD is probably your best bet, thanks to its fast response and true-color capability. Remember that the flat-panel controller must have built-in GUI accelerator.

Should the applications be graphics-intensive with a high color-information content but a lower refresh rate (e.g., instrument panels, information kiosk, factory automation, etc.), check out lower cost STN LCDs.

FUTURE TRENDS

The current flat-panel interface is complicated by the myriad signals connecting the flat panel to the video controller. But, FPDs with built-in LVDS or PanelLink interface are coming. The long-awaited FPDs with analog RGB or composite video interfacing will dominate the market by eliminating cabling, BIOS, and upgradability problems.

Increased brightness and faster response for LCDs will be achieved by using high polymer dispersion LCDs and eliminating polarizing filters. We expect ELs to be able to display full color at high resolution. EPC

Ralph Birt is codirector of engineering at Traftech, a company specializing in multimedia embedded systems. He has over 12 years' experience in embedded systems for graphical applications and is currently very much involved with MPEG solutions. You may reach Ralph at info@traftech.com or www.traftech.com.

Khoi Hoang is codirector of engineering at Traftech. Although his career began on UNIX systems for the power company of Viet Nam, he now focuses on embedded systems and multimedia applications. He has worked as an independent hardware engineer for many years. You may reach Khoi at khoih@ica.net or info@traftech.com.

SOURCES

Video Controllers

Chips and Technologies
2950 Zanker Rd.
San Jose, CA 95134
(408) 434-0600
Fax: (408) 894-2082
www.chips.com

Cirrus Logic, Inc.
3100 W. Warren Ave.
Fremont, CA 94538
(510) 623-8300
Fax: (510) 252-6020
www.cirrus.com

ATI Technologies, Inc.
33 Commerce Valley Dr. E
Thornhill, ON
Canada L3T 7N6
(905) 882-2600
Fax: (905) 882-2620
BBS: (905) 764-9404
www.atitech.ca

S3, Inc.
2801 Mission College Blvd.
Santa Clara, CA 95052-8058
(408) 588-8000
Fax: (408) 980-5444
www.s3.com

LCDs, ELs, CCFT Inverter

Sharp Corp.
Sharp Plaza
Mahwah, NJ 07430-2135
(201) 529-8200
Fax: (201) 529-8425
www.sharp-usa.com

LCDs

Toshiba America Electronics Components, Inc.
9775 Toledo Way
Irvine, CA 92718
(800) 879-4963
www.toshiba.com/taec

NEC

8 Corporate Center Dr.
Mellville, NY 11747
(516) 753-7000
Fax: (516) 753-7041
www.nec.com

FPDs

FPD Co. (USA)
2099 Gateway Pl., Ste. 100
San Jose, CA 95110
(408) 453-7007
Fax: (408) 453-6444

EL ICEBrite

Planar Systems
1400 NW Compton Dr.
Beaverton, Oregon 97006
(503) 690-1100
Fax: (503) 690-1244
www.planar.com

IRS

410 Very Useful
411 Moderately Useful
412 Not Useful

Web GUIs for Embedded Applications

With *the* advent of the Web, we now have a universal display standard. In Edward's opinion, it's just a **matter of** time until the manufacturing world also seeks a *standard* user interface, further boosting the embedded *Web server market*.

Over the last few years, the Web browser has become remarkably commonplace. You can locate data in the company database, find the names and addresses of prospective customers, and obtain current industry news.

At home, you can use it to read comics, play games, search for a job, and find out how many people in Iowa have your name.

Today, the Web browser is more common than typewriters were in the last decade. It's a de facto standard for acquiring and presenting data.

The basis of Web data presentation is the Hypertext Markup Language (HTML). This markup language is a set of functions or identifiers that define a document's look. You can determine font size, typeface, color, and position of text. It also creates tables and lists.

The HTML form is similar to some of the functions used for page layout by magazines and newspapers. Its format is similar to that of XyWrite, VAX Document, and TeX.

Along with HTML, you find Graphics Interchange Format (GIF) files, which con-

tain pictures and graphics. Many of the formats were developed for UNIX workstations, but are now standards for the World Wide Web and its browsers.

Many applications use the Web browser to display HTML and GIF images. One example is a kiosk in a museum that describes a display and links to other computers with additional information.

You'll also find that workstations or displays for manufacturing process-control systems have the familiar browser as the operator interface for process control and monitoring. Over the next couple of years, many-if not most-intelligent machines will be accessed from a Web browser.

EMBEDDED SYSTEMS ON THE WEB

Manufacturing machine controllers were some of the earliest embedded systems to be networked. But in the past, few had graphical user interfaces. The process-control industry, also early to network, had a wide variety of protocols to connect and communicate with controllers.

Most of these have migrated to TCP/IP and use coaxial cable, twisted pair, or serial lines. Today, some are even wireless.

But, most process controllers still have an old proprietary scheme to create the user interface since the GUI wasn't standardized like the network connection. So, to transfer text and graphics between computers, a number of standardized graphics databases were tried.

However, the fragmentation of the industrial market kept anyone from dominating. Enter the Internet and World Wide Web with support for personal computers.

Now, a manufacturer can build a computer-controlled system, interconnected via TCP/IP, and offer user access to the controller from any connected Web browser. All they need is an embedded kernel with an embedded Web server inside the controllers scattered across the plant.

Only a few vendors offer embedded Web servers. Most aim their product at the set-top box and embed a Web browser in the product.

Spyglass provides embedded Web servers for QNX, Phar Lap Software offers the ETS MicroWeb Server, and Agranat Systems has EmWeb for Wind River Systems' Tornado. These embedded Web servers send out HTML-formatted forms on request.

Thus any PC, network computer (NC), workstation, or other computer with a Web browser can access the data in an intelligent machine, provided the machine has an embedded Web server.

Award's WWWAccess product is an implementation of an NC or thin client using the Phar Lap ETS kernel and TCP/IP stack. The Spyglass browser that is included could be replaced with the ETS embedded Web server, thus transforming the thin client into a thin or personal Web server.

Others have similar embedded Web capabilities. In fact, if your preferred kernel doesn't have this capability, many companies provide add-ons.

Pacific Softworks offers a complete TCP/IP stack for most embedded kernels, and with the stack comes an embedded Web server. Spyglass has marketing agreements with most kernel vendors, and they too offer an embedded Web server.

It seems that the embedded-systems developer has many options. It's just that few vendors are promoting the end user's browser as the universal GUI.

EMBEDDED WEB SERVER

An embedded Web server along with hardware, kernel, and application—a so-called **Weblet**—is a self-contained, dedicated server that controls or monitors a machine or instrument. It can communicate on a TCP/IP network with any connected Web browser.

The embedded server is small enough to reside in ROM or use flash memory if more data or program space is required. Often during development, a floppy loads programs.

You don't usually think of a Web server as a ROM- or floppy-based system. The more common servers are Digital Alphas or Pentium Pros running Windows NT. But if the server is well-defined, it needn't be large or complex.

Phar Lap Software has an embedded Web server monitor-

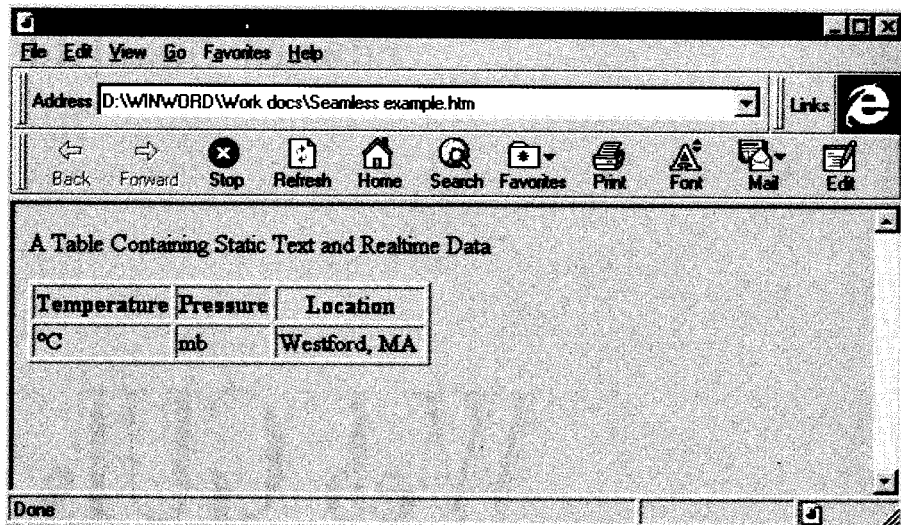


Photo 1—This screen is what a graphic artist sees on their local browser when they use the Comments fields. At this point, the Comments fields are not converted into data fields.

ing the weather outside the office building. The system consists of two PC/104 boards (3.6" x 3.8") using a 486SLC-33 and 1 MB of memory. It boots from a floppy drive and can display a dozen different HTML pages.

The system runs Phar Lap's Realtime ETS Kernel, ETS MicroWeb Server, ETS HTML On-The-Fly library, and a small application that continually reads the instruments and provides HTML pages with data on request.

HTML ON-THE-FLY

To enable the Web server to provide current data to the requesting browser, Phar Lap developed a library of C functions. The functions create an HTML page in memory, and that page includes the current data.

The library, called HTML On-The-Fly, creates HTML pages in memory and recovers the space once the page is sent to the network. The functions look similar to C printf statements with HTML tags or commands along with text and C variables.

This library lets a C programmer create a page using C language functions. The

page can contain tables, background colors and textures, and even links to other Web servers. When a requested page is defined by HTML On-The-Fly, it is created and transmitted.

If the page is undefined, the kernel looks to whatever mass storage exists to retrieve the page. This way, a developer can intermix HTML On-The-Fly pages with those created by more usual means (e.g., FrontPage).

There's one drawback to using HTML On-The-Fly as the only method of page creation. C programmers usually aren't good graphic designers. You need to mix graphic design and programming talents.

SERVER-SIDE HTML

Because of the need for graphic design, Phar Lap developed another tool—Server-Side HTML. This development aid mixes C programming and HTML page layout.

Server-Side HTML permits a graphic designer to completely create the look and feel of the user interface and still be able to insert real-time data on request. The designer creates a page using all the typical tools.

Except, where real-time data is supposed to appear, the designer inserts a Comment containing the name of the C program variable, as you see in Figure 1. When this HTML page is placed on the storage media of the embedded Web server and requested by a browser, the HTML On-The-Fly functions insert the current data stored in the named variables.

```
<P> A Table Containing Static Text and Real-time Data</P>
<Table Border>
<TR>
  <TH>Temperature</TH>
  <TH>Pressure</TH>
  <TH>Location</TH>
</TR>
<TR>
  <TD> <!-- Field temp_deg_C --> &#176;C </TD>
  <TD> <!-- Field pressure-mb --> mb </TD>
  <TD> Westford, MA</TD>
</TR>
</Table>
```

Figure 1—This HTML code includes real-time data using the special Comments fields. The Comments are enclosed with brackets and start with an exclamation mark.

Designers can see what the page looks like without data fields or they can insert dummy values. The HTML code in Figure 1 produces the outputs shown in Photos 1 and 2.

These development tools let programmers build the application and control data collection while graphic designers create the GUI displaying the data.

BIDIRECTIONAL COMMUNICATIONS

This isn't a one-way street. Data can be sent from the browser to control the process connected to the embedded system.

All Web browsers let data be included with the URL sent to the Web server. This same function asks the Web server to search for data or can select which data to display.

The Phar tap weather station lets you request data in metric or English units or displays peak wind gusts and average temperatures for the day.

Using the Web browser with a state-of-the-art real-time embedded kernel, embedded Web Server, and tools to automatically format real-time data, the intelligent machine can be accessed by anyone on the network with access privileges. Or, it can be made available to the Web like the Phar tap weather station.

NO PLUG-INS REQUIRED

Most Web servers, including embedded Web servers, permit Java or ActiveX plug-ins to display data dynamically. But, they aren't needed with the tools I described.

In fact, you may use H/PCs (hand-held personal computers) or NCs and may not have the memory to store a plug-in or the connection to download one. Plug-ins are nice and can be used in conjunction with these HTML tools, but they're unnecessary.

Dynamic HTML—a new way to display HTML forms—is currently being defined and seems to replace Java and ActiveX. It runs only on the browser and cannot pull data from the embedded system.

"FREE" DEVELOPMENT

Most new technologies come with a high price tag. Not so if you use HTML. Nearly everything needed to implement a **Weblet** using a Web browser is already in place.

You need a development system with an embedded real-time kernel and a TCP/IP stack to network embedded computer products. An embedded Web server is needed,

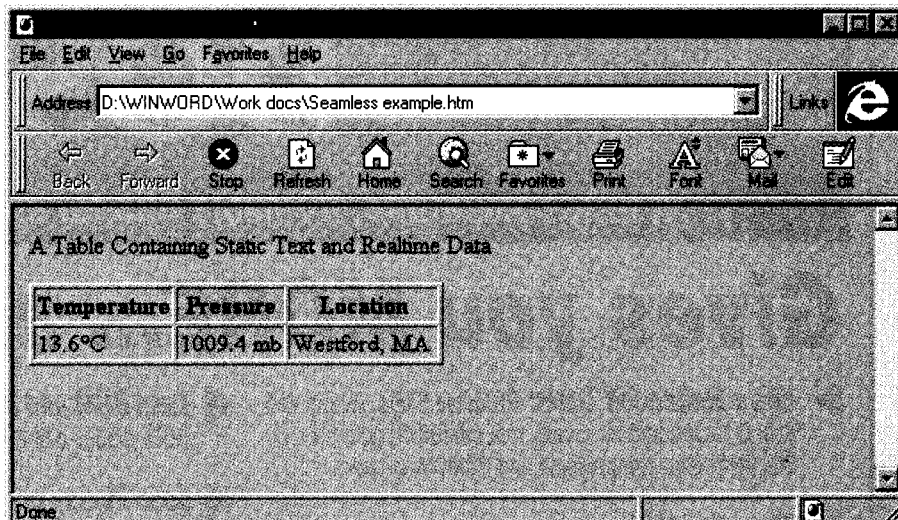


Photo 2—Here's what the user sees on their browser when connected to a **Weblet** using the Server-Side and HTML On-The-Fly technology. Unlike Photo 1, these **data** elements contain real-time data fields.

but the Phar tap **MicroWeb** Server is free with the Realtime ETS Kernel. (Some vendors may charge for their embedded Web server.)

The graphic artist most likely has **FrontPage** or some other HTML editor, and the customer has or can obtain the Web browser free. No new purchases are required.

A Web browser is a natural user interface to intelligent machines—not because it's easy to use but because it's everywhere. With tools such as HTML On-The-Fly and Server-Side HTML, the development of the machine's look and feel is easy and easily integrated into the product. **EPC**.

Edward Steinfeld has more than 25 years' experience in real-time and embedded computing. He has marketed embedded and real-time products to OEMs and resellers for Digital Equipment Corporation, Ventur-Corn, and Phar lap Software. You may reach him at stein@ma.ultranet.com.

SOURCES

EmWeb

Agranat Systems, Inc.
1345 Main St.
Waltham, MA 02 154
(617) 893.7868
Fax: (617) 893.5740
www.agranat.com

WWWAccess

Award Software Intl.
777 E. Middlefield Rd.
Mountain View, CA 94043
(415) 968.4433
Fax: (415) 526.2392
www.award.com

Axis Communications
4 Constitution Way, Ste. G
Woburn, MA 01801
(617) 938-1 188
Fax: (617) 938-6161
www.axisinc.com

Integrated Systems, Inc.
201 Moffett Park Dr.
Sunnyvale, CA 94089
(408) 542-1 500
Fax: (408) 542-1 956
www.isi.com

Microware Systems Corp.
1900 NW 1 14th St.
(5 15) 223.8000
Fax: (5 15) 224-1 352
www.microware.com

TCP/IP stack
Pacific S&works
4000 Via Pescador
Camarillo, CA 93012
(805) 484-2 128
Fax: (805) 484.3929
www.pacificsw.com

ETC **MicroWeb** Server, Real-time ETS Kernel, HTML-On-The-Fly, Server-Side HTML

Phar Lap Software, Inc.
60 Aberdeen Ave.
Cambridge, MA 02 138
(617) 661.1510
Fax: (617) 876-2972
www.pharlap.com

QNX Software Systems
175 Terence Matthews Cres.
Kanata, ON
Canada K2M 1 W8
(613) 591-0931
Fax: (613) 591-3579
www.qnx.com

Embedded Web servers
Spyglass, Inc.
1240 E. Diehl Rd.
Naperville, IL 60563
(630) 505-1 010
Fax: (630) 5054944
www.spyglass.com

Tornado
Wind River Systems
1010 Atlantic Ave.
Alameda, CA 945 10
(5 10) 748-4 100
Fax: (5 10) 8 14-2010
www.wrs.com

IRS

- 413 Very Useful
- 4 14 Moderately Useful
- 4 15 Not Useful

Video on the PC/104 Bus

Capturing and displaying live video is an action-packed arena. It calls for high processing speed and instant throughput. Richard reviews the basics of video, offering suggestions of how to get video through a very bottlenecked bus.

Capturing and displaying live video using a PC/104 board is similar to the old saying about stuffing a camel through the eye of a needle. The PC/104 bus just doesn't have the bandwidth to transfer live video.

The normal transfer rate for the PC/104 bus is only 5MBps. For monochrome video, the required transfer rate is 12.5 MBps. For color, it's 25 MBps.

However, there's always someone who won't take no for an answer. If the video rate is too fast for the PC/104 bus, there are alternatives—slow down the video or find an alternate bus.

In this article, I discuss three approaches that use these alternatives to work around the relatively slow PC/104 bus:

- live video is captured on a PC/104 board and transferred at a slower rate over the PC/104 bus
- when the conditions are just right, a PC/104-Plus frame grabber can transfer the live video over the PCI bus

- live video is digitized by a PC/104 board and transferred over a separate ribbon cable

But before I get into the advantages and disadvantages of these approaches, I want to define "live" video and provide a short tutorial on digital video.

LIVE VIDEO

There are many types of video—NTSC, PAL, SECAM, RS-170, S-Video, HDTV, 1024 line, noninterlaced, and line scan. When a motherboard has a built-in VGA controller, it's also called video on the motherboard. So, the word "video" can be somewhat confusing.

In this article, video means one of the commercial interlaced video formats, including NTSC, PAL, SECAM, S-Video, and RS-170. This type of video is used by a TV, VCR, or black and white TV camera.

RS-170 is a monochrome video format, while the others are color. They are all considered live-video formats because the

field rate is fast enough (50 or 60 Hz) to produce smooth motion.

Live-video formats are the most common video formats for monitoring and control applications that might require PC/104 boards. Although the line-scan video format is supported by the new ImageNation PC/104-Plus board, I don't discuss it here. I focus on getting live video into a PC/104 computer and onto the VGA display.

VIDEO TUTORIAL

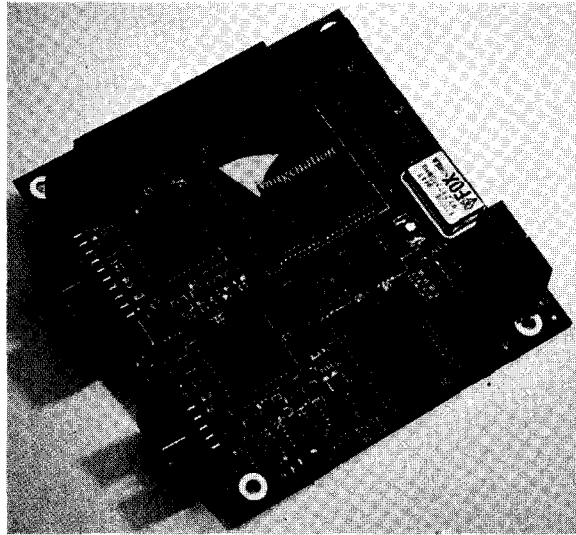
Since I live in the U.S., I'm most familiar with NTSC, S-Video, and RS-170. So, that's what I use in the examples.

People joke that NTSC stands for Never The Same Color. But, it actually stands for National Television System Committee.

The NTSC format combines color (chroma) with brightness (luminance) in a composite video signal which can be transmitted using a single video cable.

It also reduces the maximum frequency (bandwidth) of the video so that it can be transmitted over commercial TV channels.

Photo 1—
like all PC/104 basic frame grabbers, **Image-Nation's** does not support live video.



Limiting the video signal's maximum frequency has two results—reduced horizontal resolution and color instability.

If you've seen high-resolution computer graphics displayed on a TV screen, you're familiar with the blurry edges, halo effect, and chroma crawl that happen along the sharp, high-frequency edges of computer-generated text.

If your application requires better horizontal resolution, you should use RS-170 for monochrome and S-Video for color.

RS-170 is just like NTSC, except that it's black and white rather than color. There's also a very subtle timing difference (a 30 sync rate for **RS-170 vs. 29.97** for NTSC).

S-Video is similar to NTSC, except the luminance and chroma signals aren't combined. Instead, S-Video uses two video cables—one for luminance and one for chroma.

A typical S-Video cable looks like only one cable, but it actually contains two miniature coax cables. Because S-Video doesn't encode the luminance and chroma into one signal, the luminance signal can have a higher frequency, which translates into better horizontal resolution.

The relative difference is 350 visible line pairs for NTSC, compared to 550 for S-Video. RS-170 resolution is similar to S-Video. The increased resolution can be important for image processing in machine-vision applications, interactive alignment, and focusing.

Now that you're sold on RS-170 and S-Video, let's talk about the video digitizer. RS-170 and the S-Video luminance signals are analog video signals, with a higher voltage for bright areas and a lower voltage for dark areas.

An 8-bit video digitizer converts the analog voltage into a number between 0 (black) and 255 (white). A 6-bit digitizer converts the video into a number between 0 and 63, while a 10-bit digitizer converts it into a number between 0 and 1023.

The 8-bit digitizers are the most common. Usually, 256 intensity levels are enough for a human operator and for image processing. When there aren't enough intensity levels, the image starts to look posterized (like a paint-by-numbers painting).

The difference between neighboring shades of gray is clearly visible if there are too few intensity levels. You usually need 256 gray-scale steps for smooth gray scale.

In an NTSC video signal, color is encoded into a high-frequency signal that is mixed with the luminance signal. The NTSC video format is very different from how color video starts out.

Inside a color video camera, there are separate red, green, and blue signals. Like an artist's palette, the three primary colors represent all the visible colors. Similarly, a color monitor has red, green, and blue dots on the screen.

During transmission from the color camera to the color monitor, the video is usually converted to the YUV format. A straightfor-

ward transformation converts RGB (Red, Green, Blue) into YUV.

The conversion is necessary because YUV separates luminance and chroma. While RGB is a mixture of color and intensity, the Y element of YUV is just intensity. The UV elements represent only color information.

YUV's advantage is that video bandwidth can be reduced by reducing color quality alone, without affecting luminance.

The quality of YUV encoding can have the values 4:4:4, 4:2:2, or 4:1:1. The 4:4:4 encoding scheme is the best, and 4:1:1 is the worst.

The 4:4:4 designation means that for every four Y values, there are four U values and four V values. The 4:2:2 designation means that for every four Y values, there are two U values and two V values, providing half the original color resolution.

And, you guessed it. The 4:1:1 designation gives one U and one V value for every four Y values, giving only a quarter of the color resolution.

INTERLACING AND SYNC

The video digitizer outputs a series of digital values, where each value is a pixel on the scan line. The RS-170 video frame has 525 horizontal lines of video.

Only about 480 of these lines have active video, however. The others are used for synchronization.

The vertical sync pulse occurs during the inactive video lines and marks the start of a frame. Each frame has two fields—odd (lines 1, 3, 5, ...) and even (lines 2, 4, ...). The even field is transmitted, then the odd, then the even, and so on.

There is a sync pulse for each field, and the field rate is 60 Hz. Since it takes two fields to get all the lines for a complete video frame, the frame time is 30 Hz.

The even/odd field method for transmitting video

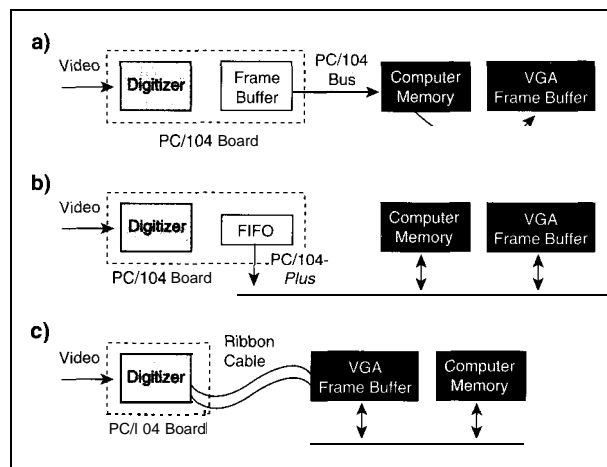
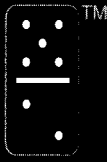


Figure 1a—The frame grabber captures a video frame, then slowly transfers it over the **PC/104** bus to computer memory. b—The frame grabber sends live video over the **PC/104-Plus** bus to the VGA frame buffer of computer memory. c—The frame grabber sends live video over a ribbon cable directly to the VGA frame buffer.

BIG THINGS COME

DOMINO
Microcontroller



DOMINO
Microcontroller
MODEL [] S/N []

Starting at
\$99

Micromint's Domino-52 microcontroller is a "supercomputer" in less than 0.75 cubic inches. We've packed the most essential elements into one tiny package. Domino is a plug-and-go module, just attach +5V and a terminal or network. A simple keyed sequence saves an autostarting program in nonvolatile memory.

SPECIAL FEATURES

- 80C52 with ROM-resident, full floating-point BASIC
- 32K bytes SRAM and 32K bytes EEPROM
- Two PWM outputs, I²C bus
- Serial I/O: (up to 19,200 bps) RS-422, RS-485 & RS-232A
- Two interrupts and three timers
- Parallel I/O: 12 bits, 3 shared with ADC and I²C
- Power: +5V @ 15 mA;
- Size: 1.75" x 1.062" x 0.4" potted
- A/D converter: 2 channels, 12 bits, 10k samples/sec.
- Connections: via 2x10, 0.1" dual-row header
- -20°C to 75°C operating temperature
- Industrial temperature available

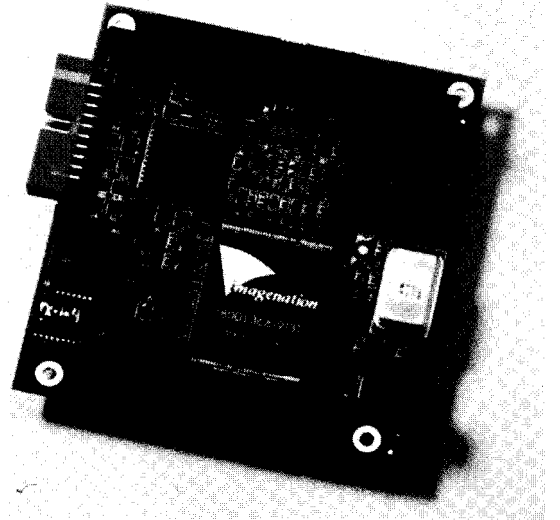


4 Park Street • Vernon, CT 06066

Call 1-800-635-3355

(860) 871-6170 • Fax (860) 872-2204

Photo 2—PC/104-Plus frame grabbers attempt to transfer live video over the PCI bus, but it's not a sure thing.



is called interlacing. By contrast, noninterlaced video transmits lines 1, 2, 3, ,....

Noninterlaced video is simpler and easier to control than interlaced video. So, why use the more complicated interlaced video in NTSC, RS-170, S-Video, PAL, SECAM,...?

The answer: our sensitivity to flicker. If the screen updates at 30 Hz, the human eye can see the update rate. At 30 Hz, moving objects appear to move in a series of small jerks.

On the other hand, if the screen updates at 60 Hz, our eyes can't detect the update rate and objects appear to move smoothly. Interlaced video gives the smooth motion of a 60-Hz update rate, while only updating the entire screen at 30 Hz.

SQUARE PIXELS

Why should you care about square pixels? For image processing and graphics, square pixels are much easier to process.

When you draw a circle using square pixels, the circle looks like a circle on screen. If the pixels aren't square (i.e., they're rectangular), the circle looks like an ellipse.

What's the big deal? Well, if you capture an image and the pixels aren't square, when you display the image on a computer monitor, the image looks squished in one direction. Also, image-processing library functions usually assume square pixels.

Let's talk briefly about how we get square pixels. By definition, a square pixel takes up the same distance horizontally and vertically onscreen.

TV screens and computer monitors are usually wider than they are tall, yielding a 4 x 3 aspect ratio. If there are 480 lines vertically, you need 640 square pixels horizontally.

Common examples of square-pixel formats are the standard computer VGA resolutions and MPEG. The standard VGA resolutions are 640 x 480, 800 x 600, 1024 x 768, and 1280 x 1024. All have an aspect ratio of 4:3.

The MPEG and MPEG2 formats are also square-pixel formats. MPEG is a 320 x 240-pixel image format, and MPEG2 has 640 x 480 pixels per image.

Two pixel rates are commonly used by video digitizers. One is an international standard, and the other creates square pixels.

A digitizer compatible with the CCIR 601 standard converts NTSC analog video to digital numbers at a 14.7-MHz rate, which equals 720 pixels along each line. In other words, 720 pixels per active line is the CCIR 601 standard.

However, the pixels aren't square. A digitizer must convert them at the 12.5MHz rate to produce 640 square pixels per line. When you buy a video digitizer, a square-pixel digitizer is best if your application requires image processing or if you intend to display the image on your computer.

Now that you're caught up on video standards, let's discuss the three possible solutions to the original problem—the fact that the data rate for live video is higher than the PC/104 bus can handle.

PC/104 FRAME GRABBER

The PC/104 frame grabber approach depicted in Figure 1 solves the bottleneck problem by slowing down the live-video data rate to a level that can be handled by the PC/104 bus.

This technique has several advantages. Installation is simple since it uses only the PC/104 bus. It's also a low-cost solution.

Its disadvantages include slow, jerky image updates on the computer monitor. During the time it takes to transfer the image over the bus, the digitizer is inactive, so

SetVideoWindowSize	SelectVideoSource
SetVideoWindowPosition	SetVideoFormat
FreezeVideo	SetColorKey
UnFreezeVideo	SetZoomFactor
SetHue	SaveImage
SetBrightness	LoadImage
SetContrast	

Table 1-These **typical software library** functions are basics in almost any **imaging** library.

several frames are skipped. As well, the slow image-transfer rate adds latency to the image-processing time, which can be critical in real-time process control.

In addition to PC/I 04 boards from ImageNation (see Photo 1), devices like Snappy from Play Inc. and QuickCam from Connectix are in this category.

The Snappy module captures the image in pieces, so it can take several frames to grab the entire image. The Snappy video-capture module and QuickCam plug into the parallel port and operate similar to the PC/I 04 frame grabber.

Their image update rate is slow (several times per second), but their cost is low. Snappy and QuickCam sell for less than \$200.

PC/104-PLUS FRAME GRABBER

If the PC/104 bus is too slow, try the new, wonderful, high-performance PCI bus. You just need to repackage it to make it more rugged and then give it a new name, like PC/I 04-Plus or CompactPCI.

If I seem insincere about the wonderful PCI bus, it's because I've built products that use the PCI bus for live video. It can be done, it just isn't easy or reliable.

It does have its advantages, however. Installation is simple, since it uses only PC/104-Plus (see Figure 1 b). It provides fast, real-time image transfer to the host memory or VGA frame buffer. And, its low transfer latency speeds up image-processing time.

However, the PC/I 04-Plus bus is not always fast enough to transfer live video, and bus contention can corrupt image data.

Live image transfer bandwidth can reduce the performance of other peripherals, like disk drives, VGA controller, and Ethernet. And, it costs more than other solutions.

Shown in Photo 2, ImageNation's PC/I 04-Plus frame grabber is a sophisticated product that supports other video formats besides live video. However, its performance depends on the PCI bus.

Not all PCI buses are the same. For example, the early PCI bus on the Intel

Neptune motherboard couldn't support live-video transfer rates. Before committing to a design, test the PCI-bus performance under the maximum anticipated load.

Let me get on my soapbox about the PCI bus. And since the PC/104-Plus and CompactPCI bus are just different physical implementations of the PCI bus, my tirade also applies to them.

I think the PCI bus is a great design for a multitasking operating system with multiple bus masters. Its arbitration scheme guarantees that each bus master has its chance to transfer data over the PCI bus.

This system is great when you have several devices all trying to use the PCI bus. The devices may include the disk controller, Ethernet, and the CPU writing to the VGA controller.

The PCI-bus arbiter is usually programmed to let each device perform four consecutive long-word transfers (i.e., four transfers of four bytes each, totalling 16 bytes).

After four transfers, a bus master is kicked off the bus, and a different bus master is allowed to access the PCI bus. This feature is great for multitasking but bad for transferring video.

Each image is a large block of consecutive data. It would be better if the PCI frame grabber took control of the PCI bus and transferred at least several thousand bytes of data before releasing the bus.

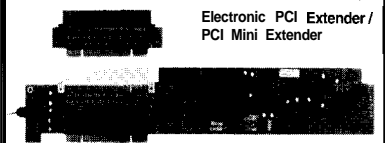
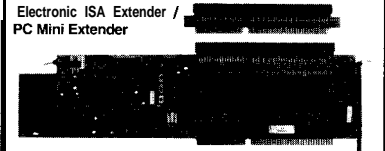
The overhead required to reacquire control of the PCI bus after only four transfers reduces the average transfer rate. In spite of the 132-Mbps maximum burst transfer rate, four cycle bursts make the PCI bus marginal for transferring live video.

Another problem with the PCI bus is contention with other devices. Since the frame grabber is releasing the PCI bus after four cycles, other devices can use a significant amount of the PCI bandwidth, causing an incomplete image transfer.

In a typical PCI frame grabber, the video is digitized and sent through a FIFO to the PCI-bus interface. The FIFO is a small buffer that allows the frame grabber to wait for its turn on the PCI bus.

IF YOU DO
FUNCTIONAL TESTS
YOU NEED
HOT SWAPPING ELECTRONIC EXTENDERS

For PCI, ISA, VXI, EISA, VESA, Micro Channel & NuBus. Custom Designs available.



- Insert/Remove Cards With PC Power On!
- Save Time Testing And Developing Cards
- Save Wear On Your PC From Rebooting
- Adjustable Overcurrent Sensing Circuitry
- NO Fuses, All Electronic For Reliability
- Single Switch Operation W/Auto RESET
- Optional Software Control Of All Features
- Breadboard Area For Custom Circuitry
- And More...

Full line of passive extenders:
PCI, ISA, VXI, EISA, VESA,
Micro Channel & NuBus

Passive PCI Extender Passive EISA Extender

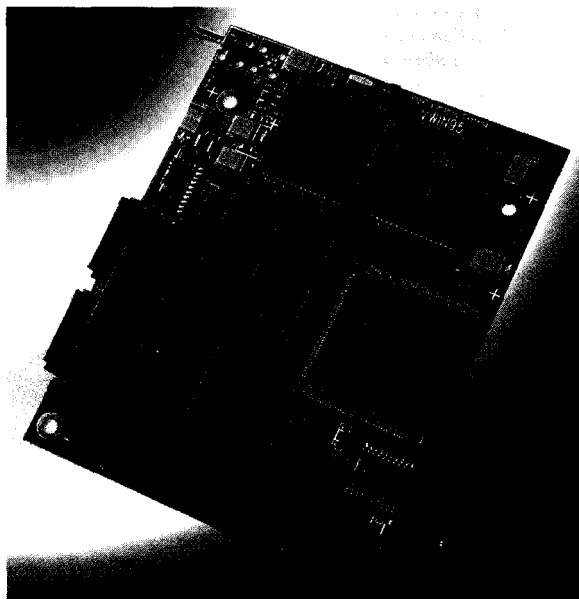


Passive MC32 Extender Passive ISA Extender



AZ-COM, INC.
3343 VINCENT RD., #D
PLEASANT HILL, CA 94523 U.S.A.
TEL: 510-947-1000
FAX: 510-947-1900
24-Hour Fax on Demand:
510-947-1000 Ext. 7
Email: sales@az-com.com
VISIT OUR HOME PAGE AT:
<http://az-com.com/>

Photo 3—
This PC/104 frame grabber from Hopkins Imaging Systems transfers live video over a ribbon-cable video bus.



When the frame grabber is kicked off the bus, image data starts to accumulate in the FIFO. When the frame grabber regains control of the bus, it has to quickly empty the FIFO by sending data over the PCI bus. If it doesn't have enough time, the FIFO overflows and image data is lost.

When a PCI frame grabber is using most of the PCI bandwidth, other devices (e.g., the disk drive, Ethernet, and the VGA controller) have reduced access to the PCI bus, which can noticeably reduce performance.

RIBBON-CABLE VIDEO BUS

The third approach to the PC/104 bottleneck is to design a ribbon-cable video bus as illustrated in Figure 1c. While this solution sounds a bit crazy, it's worked so well that it has become an international standard.

Its advantages include guaranteed real-time image transfer to VGA frame buffer. As well, its low transfer latency speeds up image processing time. It's also low cost.

The only disadvantage is that an additional ribbon cable is required for installation.

The ribbon-cable video bus includes both old and new video buses—the PC Video and Zoom video buses, respectively. The Zoom video bus is expected to replace the PC Video bus and is included in most new Toshiba portable computers.

There are PCMCIA cards available from Toshiba and Margi Systems that enable a Toshiba portable computer to capture and display live video-in-a-window on the flat-panel display.

PC Video frame-grabber cards are available for the Ampro, Octagon, and Adastra SBCs. Adastra makes a PC Video frame grabber for their SBC, and Hopkins Imaging Systems [see Photo 3] builds PC Video

frame grabbers for the Adastra, Ampro, and Octagon SBCs.

PC Video frame grabbers are more expensive than the new Zoom video frame grabbers (about \$700, compared to less than \$300). A Zoom video frame grabber is currently available from Hopkins Imaging Systems for the Octagon SBC and will soon be available for the Ampro SBC and PC/104 VGA controller.

DON'T FORGET THE SOFTWARE

Software libraries make it easy to integrate the frame-grabber and image-processing functions into an embedded PC/104 computer system. Some typical functions are listed in Table 1.

Adastra, Hopkins Imaging Systems, and ImageNation provide software libraries with their products. Snappy and QuickCam include software libraries only on special request.

MAKING CHOICES

It's funny that the phrase "pie in the sky" rhymes with PCI. The PCI bus is a wonderful, high-performance bus that can solve all your problems—unless you need live video.

Even a monochrome image is a large (300 KB) block of data, and the PCI bus isn't the best design for transferring large blocks of data. The problems with the PCI bus apply to the PC/104-Plus, CompactPCI, and other PCI-bus incarnations as well.

So, if you don't need fast image transfers, get a low-cost PC/104 frame grabber or one of the low-cost parallel port devices. If you need reliable live video-in-a-window

or fast image capture, a low-cost Zoom video board may be your answer. PCQ.EPC

Special thanks to Thomas W. Hinckley of Studio 1501 Photography for the cover photo for the Embedded PC section and Photo 3.

Richard Hopkins is a programmer, system engineer, and president of Hopkins Imaging Systems. He has worked with flight simulators for the military, developed image processing software for the special-effects industry, and designed embedded-PC systems for production-line inspection and x-ray imaging. You may reach Richard at rhopkins@hopkinsimg.com.

SOURCES

PC Video boards
Adastra Systems
26232 Executive Pl
Hayward, CA 94545
(5 10) 732-6900
Fax: (5 10) 732.7655

PC/104-Plus

Ampro Computers, Inc
990 Almanor Ave.
Sunnyvale, CA 94086
(408) 522-2100
Fax: (408) 720-1305

QuickCam

Connectix Corp.
2655 Campus Dr.
San Mateo, CA 94403
(415) 571-5100
Fax: (415) 5715195

Zoom video board, PC Video boards
Hopkins Imaging Systems
18 12 Flower Ave.
Duarte, CA 91010
(8 18) 305-8833
Fox: (818) 305.8838

PC/104 frame grabber

ImageNation Corp.
P.O. Box 276
Beaverton, OR 97075-0276
(503) 64 1-7408
Fax: (503) 643.2458

Frame grabber
Margi Systems, Inc.
3 155 Kearney St., Ste. 170
Fremont, CA 94538
(5 10) 657.4435
Fax: (5 10) 657.4430

SBC
Octagon Systems
6510 West 91 Ave
Westminster, CO 80030
(303) 430-1500
Fax: (303) 429-8126

Snappy

Play Inc.
2890 Kilgore Rd
Rancho Cordova, CA 95670
(916) 851-0800
Fax: (916) 851-0801

IRS

4 16 Very Useful

4 17 Moderately Useful

4 18 Not Useful

Applied PCs

Fred Eady

Internet Appliance Development

Part 2: Getting Flow-Meter Data

Fred uses Photon, SLANG, and EXPLR2 to remotely collect flow-meter data. After looking at the flow meter's encoding method, you learn how to read, process, and transform *the data* for month-end invoices—all via the Internet.

I'll never be lonely. As long as I live here, I'll welcome at least three visitors a month.

One reads the gas meter. Another reads the electrical meter, and the third collects the water digits. Depending on your geographical location and the budgets your utility companies live on, you may also get a monthly visit from my three friends.

On the other hand, some of you may be forever without a meter-reading pal. The human touch of the utility-meter-reading trade is gradually being replaced by—you guessed it—embedded computer technology.

TOOLS OF THE TRADE

In Part 1, I detailed what my embedded platform is (see Photo 1), what it can do, and what software is available in the context of this application. At this point, the highest

priority in the project is procuring a suitable sensor (i.e., meter).

When I find myself in an engineering tight spot, it's process-of-elimination time.

First of all, I don't have a natural gas port in the shop. So, I won't need to find a cubic-foot flow meter. I hate working with

high voltages. So, I won't need an electrical watt-hour meter, either.

Getting wet doesn't bother me too much, and besides, it's safer working with cold water than explosive gases and lethal voltages. So, ladies and gentlemen, a water-meter project it is!

As it turns out, the water choice works out great. Gee, I just happen to have a water-flow meter on the bench. Where'd that come from?

NEPTUNE ARB V

Does that name exude wetness or what? The Neptune ARB V is designed for commercial water-flow measurement environments.

The ARB V allows automated remote reading, enabling the billing process to be automated as well. As you see in Photo 2, it's a sturdy, well-engineered piece of

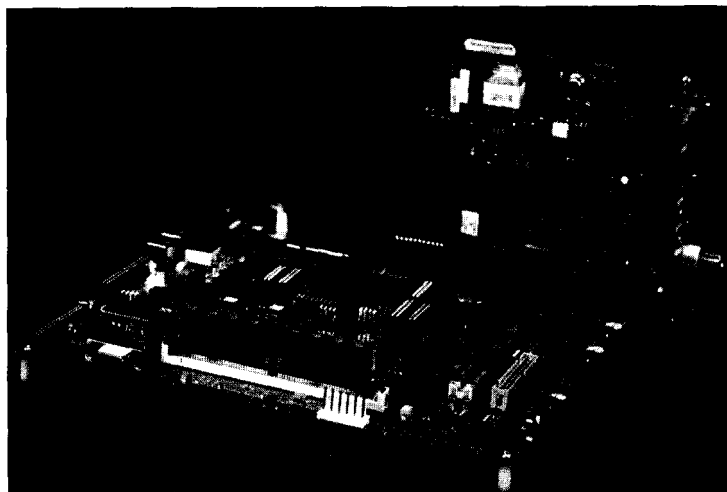
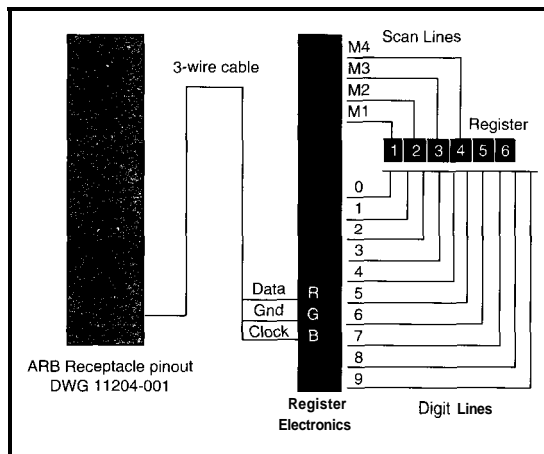


Photo 1—The EXPL2 footprint is larger than most of the evaluation boards I've used, but there's lots of hardware on that chunk of circuit board....

Figure 1—The ARB Vuses special register electronics to replace the older 14-wire technology.



equipment that's designed to be put somewhere you and I wouldn't wanna be and left there alone for a very long time.

In its native implementation, the ARB V is coupled with a remote receptacle that provides location and identification information. The remote receptacle is an intelligent device capable of storing customer ID records and ARB configuration data.

This receptacle can also temporarily store ARB-generated data for later retrieval. I don't plan to install an ARB in every home on my block, so my "remote receptacle" and accompanying data will be replaced by the EXPLR2 embedded PC and SLANG.

In normal day-to-day operation, meter data from the ARB V is collected by specialized reading equipment and stored in some sort of memory device. After all the daily readings are done, the collected data is offloaded into a central computing device for processing.

Traditionally, data collection is done via human hands with special probes or automatically via RF or telephone facilities. I'm going to break tradition and add another means of viewing the ARB V's data—the Web page.

LET'S GET WET

My ARB V has a six-wheel encoding option for very precise readings. For my model, the least significant digit (LSD) represents 50 gal. Similar model four-wheel ARBs are available, with LSD readings equaling 1000 gal.

Each wheel implies a readable digit. Thus, four-wheel meters display and transmit four-digit readings, while six-wheel meters produce readings with two additional LSDs.

Synchronizing the inside and outside registers is accomplished by the ARB V "reading itself" and then transmitting the reading displayed on the outside register odometer. A three-wire data transfer connection links the ARB V meter and a respec-

tive remote receptacle. Using 22 AWG, the ARB V can be positioned up to 300 away from its remote receptacle.

Three-wire reading is a relatively new technology that replaces 14-wire reading. Electronics were added to the ARB V internal register to provide a scanning function of the number wheels. Figure 1 depicts the internal register electronics.

The three wires connected to the register are Clock/Power, Ground, and Data. Providing a clock signal to the register and reading the synchronous serial data from the register performs register data acquisition. Power for the internal register reads is supplied via the clock/power pin.

Basically, a capacitor captures enough charge from the incoming clock to power the internal register's electronics. The maximum applied clock frequency cannot exceed 5 kHz for proper operation.

NEPTUNE'S TRITON

Each bit of data retrieved from the ARB V comprises four clock phases. Thus, each

data byte read from the ARB V is the result of 32 clock cycles.

As Figure 2 shows, each bit is a unique set of levels with respect to the clock phases. Notably, the least significant bit (LSB) phases of each character differ from the remaining most significant bit (MSB) phases.

This scheme lets the receptacle programmer sync to the beginning of a meter digit. Once the sync point is established, 256 clocks are applied that result in the gathering of 16 data nibbles containing the meter reading. These 16 nibbles compose the ARB V data word.

The ARB V data-bit sync point is found by toggling the ARB V clock line high to low and reading the data line. This continues until the data line is found to be low following the high-to-low clock transition.

Once this low-clock/low-data condition is met, the clock is toggled high. As a result, the data line should go high as well. This sequence is represented in Figure 2 as the short pulse occurring just prior to phase 1.

Note that, during phase 1, the LSB level patterns are both high and the MSB patterns are both low. This level pattern is unique to phase 1.

The receptacle programmer simply loops the pulse-pattern routine until the phase-I condition is met. This method establishes a beginning sync point regardless of where you enter into the internal-register data-phase patterns.

Once the sync point is established, the entire data word (16 nibbles) is read. Figure 3 illustrates the entire ARB V data word.

For the six- and four-wheel ARB Vs, the meter reading begins at the seventh nibble

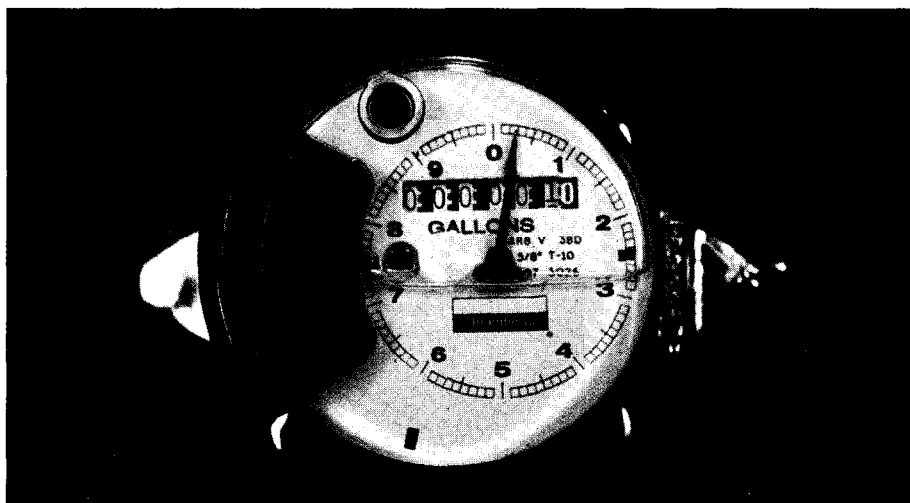


Photo 2—As if this device wasn't sturdy enough, it's filled with oil for long-term reliability.

and ends at the tenth. The seventh nibble is the most significant digit (MSD).

A six-wheel ARB V posts data in nibbles five and six. A four-wheel ARB V registers hex E (binary 1110) in these positions. Interpret E as the digit 0 (binary 0000).

With respect to the six-wheel ARB V, the fifth digit is fully encoded and the sixth digit is really a half digit. If valid data is returned in digit six, it can only be 5 or E. The E in the sixth digit is defined as 0 if the fifth digit is 0-9 or B.

So, if you don't know how many wheels your ARB V has, first decode the fifth and sixth digits and look for valid values. If valid digits are found in the fifth position, then any E in the sixth position is decoded as 0.

A 5 in the sixth digit and an E in the sixth digit indicate an error in that digit. Es in both positions indicate a four-wheel ARB V. Table 1 lays out who can be where.

Let's assume the following data was acquired from a six-wheel ARB V:

EE EE EF EE EE 4E 21 99

Notice that the nibbles are out of place with respect to well-mannered Figure 3. To

Character No.	Meter Digit	Allowable Hex Values
a)	7 MSD	0-9, B, E
	8 2nd MSD	0-9, B, E
	9 3rd MSD	0-9, B, E
	10 LSD	0-9, B, E
b)	7 MSD	0-9, B, E
	8 2nd MSD	0-9, B, E
	9 3rd MSD	0-9, B, E
	10 4th MSD	0-9, B, E
	5 5th MSD	0-9, B, E
	6 LSD	5, B, E

Table 1a—Here's the lowdown on the digits versus registers for the four-wheel encoding. b—Note that, in the six-wheel encoding, digit 6 can only be 5 or 0.

get accurate readings, we must establish a beginning sync point.

It's possible to enter the algorithm in the middle of a phase group. That phase group might be nibble x of the 16 nibbles. No matter where we enter the data packet, we still read 16 nibbles before we're done beginning at the front of one of the nibbles.

From the example data, it's obvious that the reading is contained within the numeric digits. But, how do we get the datastream to correspond with Figure 3's layout?

The answer lies in the lone F in the datastream. F is an end-of-word indicator.

To decode the meter reading, first find the F in the data word. Go forward six nibbles. This move places you at the E just right of the 4.

The reading starts at the seventh nibble (i.e., 2) and continues for four positions. The LSDs are contained in the fifth and sixth nibbles forward from the F.

So, the reading would be 21994E with E decoding to 0. You can also rearrange the reading to place the F at the end of the data word and use Figure 3 to find the reading, which results in the pattern:

EE EE 4E 21 99 EE EE EF

Any Bs or Es other than the E in the 4E byte gives an error condition. A B indicates a short, and E points to an open condition.

Let's talk more about error conditions. Two types of errors can occur in the three-wire data. These are errors in the transmitted data or errors indicated by the data.

Indicated errors are opens and shorts. Transmitted data errors present themselves



Looking to fill your embedded toolbox without breaking the bank???

DDS COMPLETE

Everything you need, all in one convenient package! Includes all of our Micro-C Compilers, Assemblers, Simulators, Disassemblers, Monitors, Project Plans, and Data Line Monitor. Many CPUs supported, including C-FLEA, a virtual CPU you can use anywhere.
Package price: \$ 500 US (\$ 650 CDN).

*Call or write for our free catalogue .
FREE Demos available on BBS or WEB (or send \$5 for diskette)
We accept Cheque/MO/Purchase Orders/VISA*



Dunfield Development Systems


P.O. Box 31044 Nepean, Ont. K2B 858 CANADA
Tel: 613-256-5820 Fax: 256-5821 BBS: 256-6289
info@dunfield.com, sales@dunfield.com,
http://www.dunfield.com

Industrial Strength!

SINGLE BOARD COMPUTER

The EPAC 3000G2 stands up to harsh environments and has the Power to handle tough, demanding applications. With Serial, A/D, D/A, LCD, Keypad, Real Time Clock, Dipswitch, EEPROM, and Flash. \$379 Qty. 1

- 8 Optically-Coupled Inputs & 20 Programmable Digital I/O Lines
- 8 Optically-Coupled High-Drive Digital Outputs
- 16 Channels of 12/16 Bit A/D & Opt. 2 Channels of 12 bit D/A
- 2, 16 bit Timers & 4, 8 bit Counters
- Up to 5 RS232/485 Serial Ports
- Backlight Capable Character & Graphic LCD Interfaces
- Optional 16 Key Keypad & Interface
- 1 Meg of Memory Space Total. 64K ROM & 128K RAM Included
- **Assembler & Monitor Included, BASIC, C, and Forth Optional**



1985 - 1997
OVER
12
YEARS OF
SINGLE BOARD
SOLUTIONS

EMAC, inc.

618-529-4525 Fax 457-0110 BBS 529-5708
11 EMAC WAY, CARBONDALE, IL 62901
WORLD WIDE WEB: <http://www.emacinc.com>

The PC/104 Motion Control Experts

Need motion control within your PC/104 application?

Overwhelmed by the number of products & vendors out there? Looking for a motion control specialist instead of just another PC/104 vendor with

one motion controller?

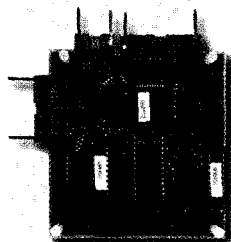
Look no further. With more than a decade of experience supporting the motion control needs of OEM customers, Tech 80's family of



Model 5912

Encoder Interface

PC/104 modules can meet the encoder interfacing and servo & stepper control demands of your embedded application.

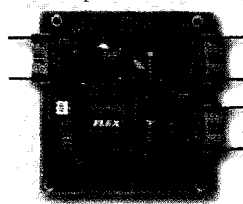


Model 5928

Servo Controller

All Tech 80 products feature extensive C-compatible software libraries, example code and tools to provide you with a flexible development environment.

AND if your needs extend beyond the PC/104 realm, Tech 80 has the industry's most extensive line of board-level motion control products for IP, PC, STD and VME-based systems.



Model 5936

Stepper Controller

For more information on our PC/104 motion control products or to speak with an sales engineer regarding your current project,

please contact us at 800/545-2980 or visit us at:

www.tech80.com/cc/pc104.html



The People in Control of Motion™

Minneapolis, Minnesota USA
800/545-2980 . 612/542-9545 . 612/542-9785 (fax)
www.tech80.com . info@tech80.com

listing I-Here's how you can employ SLANG to implement a PPP connection.

```
#!/usr/cogent/bin/slang /* Talk to serial port */
defvar (modem, "/dev/ser2");
defvar (baudrate, 57600);
defvar (verbose, nil);
defvar (phone, "4169554350");
defvar (logprompt, "login:");
defvar (passprompt, "password:");
defvar (timeout, 60);
defvar (pppdummy, "dummy.company.com");
defvar (logname, "Pname");
defvar (password, "password_here");

function do_login(fd, phone, logprompt, logname, passprompt,
password, timeout) {
    local status;
    dev_write (fd, "atzL3\r");
    if (waitfor (fd, "OK", timeout, verbose)) {
        dev_write (fd, string ("atdt", phone, "\r"));
        if (waitfor (fd, logprompt, timeout, verbose)) {
            dev_write (fd, string (logname, "\r"));
            if (waitfor (fd, passprompt, timeout, verbose)) {
                dev_write (fd, string (password, "\r"));
                status = t; }}}
    status: }

function waitfor (fd, chars, timeout, echo) {
    local len, buf, inch, done, newlen, status;
    len = strlen (chars);
    buf = "";
    for (; !done;){
        inch = dev_read (fd, 1, 1, 0, timeout * 10);
        if (inch != ""){
            if (echo) {
                princ (inch);
                flush (stdout);}
            buf = string (buf, inch);
            if (strstr (buf, chars) < 0){
                newlen = strlen (buf);
                if (newlen > len + 32)
                    buf = substr (buf, strlen (buf) - len - 32, -1);
                else done = status = t; }
            else done = not (status = nil); }
    status: }

function read-to (fd, char, timeout) {
    local inch, buf, done;
    for (buf=""; !done;){
        inch = dev_read (fd, 1, 1, 0, timeout * 10);
        buf = string (buf, inch);
        if (inch == char)
            done = t;
        else if (inch == ""){
            done = t;
            buf = nil; }}
    buf: }

function get_ip_address (fd){
    local buf, buf2;
    if (buf = read-to (fd, " ", 20)) {
        if (strchr (buf, "(") >= 0)
            buf2 = read-to (fd, ")", 20);
        else
            buf2 = read-to (fd, " ", 20);
        if (buf2)
            buf = car (string-split (string (buf, buf2), "()", 1));
        else
            buf = nil; }
    buf: }

function main 0 {
    local fd, gateway_ip, local_ip, have-modem, have_login;
    for (i=cdr(argv); i: i=cdr(i)){

```

(continued)

Listing 1-continued

```

if (car(i) == "-v")
    verbose = t;
else if (!have_modem){
    modem = car (i);
    have_modem = t; }
else if (!have_login){
    logname = car (i);
    have_login = t; }
else
    password = car (i);}
fd = dev_open (modem, 0);
dev_setup(fd, baudrate, 8, 'none', 1, 0, 0);
if (do_login(fd, phone, logprompt, logname, passprompt,
    password, timeout)) {
    system (string ("stty +hupcl < ", modem));
    system (string ("pppd defaulttroute",modem,"",baudrate,"&"));
    sleep (5);}
else
    princ ("Login attempt timed out\n");}

```

as illegal characters. Usually, reading the ARB V while a digit is transitioning causes an illegal character. To avoid this kind of error, the ARB V should be read until two consecutive readings are identical.

The first reading is taken with a 50% duty-cycle 3.2-kHz clock. The second reading is taken with a stretched clock.

The period of the low portion of the clock is retained, while the high portion is

extended to a 66% duty cycle. The resulting clock for the second read is 2.133 kHz. A no-data situation indicates that the internal electronic register is malfunctioning or the meter being read is a 14-wire type.

TROUBLED WATERS

OK, time to put the spurs to the ARB and get some data. The EXPLR2 parallel port is a perfect candidate for our ARB I/O-not!

Listing 2-Looks and feels a lot like C, huh?

```

#!/usr/cogent/bin/slang
mkdir("/tmp".0o777); //make sure there is a /tmp dir:
read-data = make-array(1);
function read-pic 0 {
    local read-data,i;
    read-data = make-array(0);
    for(i=0;i<2;i++) {
        // port/PIC handshake code goes here }}

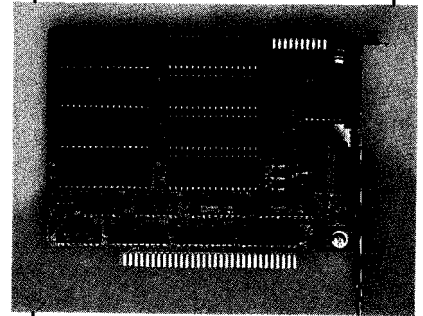
function write-data-to-file 0 {
    local fp,i;
    if (fp=open("/tmp/water_data.html","w")){
        writec(fp,"<HTML><HEAD>H2O Meter Readings</HEAD>");
        writec(fp,"<BODY><H1>Latest Water Meter Readings</H1>");
        writec(fp,"<EM>Last Updated: ",date(),"</EM><PRE>");
        writec(fp,"Meter Reading\n");
        writec(fp,"---- -\n");
        for(i=0;i<2;i++) {
            writec(fp,format("%-5d %-7d\n",i,read-data[i])); }
        writec(fp,"</PRE></BODY></HTML>");
        terpri(fp);
        close(fp);
        spawn-ftp(); } }

every(360,#read-pic());
every(3600,#write-data-to-file());
while (t){
    next-event(); }

function spawn-ftp 0 {
    qnx_spawn-process(nil, 0, -1, -1,
    _SPAWN_BGROUND & _SPAWN_NOZOMBIE,
    "/usr/bin/ftp",
    list("webhost.domain.com"),nil, -1); }

```

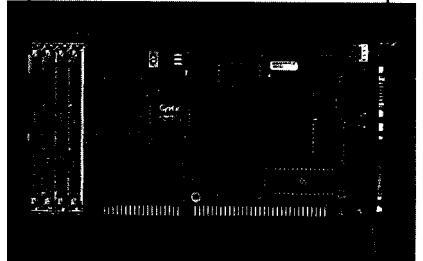
VMAX[®]
 QUALITY PRODUCTS
 RESPONSIBLE SERVICE
 RELIABLE DELIVERY



ENHANCED SOLID STATE DRIVE — \$89.00
 5 Flavors: 2 or 3 Drives, 2M,4M, 32M, 2M with Post LEDs
 Either Drive Boots, FFS included
 112 Card, XT Height, Customs too



MULTI PORT I/O BOARD — \$90.00
 4 Serial Ports, 2 Bi-Parallel Ports, 2 Drive IDE, 2 Floppy Interface, 16 Bit Standard, 8 Bit Adaptable, Low Profile 4.2", Full Cable Set



'386 66MHz SINGLE CARD COMPUTER — \$335"
 Up to 2.5MegFlash/Sram drive
 Compact-XT height 1/2 card size
 Industry Standard PC-I 04 port
 L.2 cache to 64K—DRAM to 16Meg
 Dual IDE/Floppy connectors
 All TempustechVMAX® products are FC 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: (941) 643-4981
 E-Mail: cpusales@tempustech.com
 I-Net: www.tempustech.com

Fast for 295 Airport Road
 fast response! Naples, FL 34104

It seems there's a slight problem. No, the hardware is fine. No, it's not the software, either. This combination of hardware and software for a "standard" Web application is great. Problem is, this ain't no standard Web application.

Think back. Remember that one of SLANG's claims to fame is bumpless updating of executing SLANG code? By mixing SLANG with specially compiled modules from other languages, we severely complicate and possibly impair this application.

In other words, yes we can use SLANG and other hardware. But, that defeats our purpose. Can we do this with SLANG and the EXPLR2?

What if I want to update the elapsed time between readings or change the way the bits from the parallel port are read? In this hardware environment, SLANG can do part one, but part two is a different story.

The problem doesn't lie in SLANG, and I can't put down the Intel '386EX, either. The truth is, coding exclusively with SLANG, the EXPLR2's 33-MHz clock speed is too slow to accommodate the minimum clock frequency needed by the ARB V. That's without factoring in the time required to read and store the ARB phase patterns.

I generated the clock code with SLANG on a QNX-equipped '586 133-MHz PCI system. The loop argument was 28 decimal to obtain the 2.133-kHz 50% duty-cycle clock without reading the incoming bits.

In that it's not prudent to equate megahertz with processor cycles, we can be assured that dividing the 28 and multiplying accordingly by 33 MHz soon puts us out of business (intuitively and physically) as far as ARB V timing loops are concerned.

If that's not enough to snuff this project, the evaluation system isn't a full implementation of QNX. There's just enough QNX and SLANG on the EXPLR2 to run the demos.

DO WATER SPIDERS SPIN WEBS?

If we're gonna get the readings to a Web page, here's what's gotta happen.

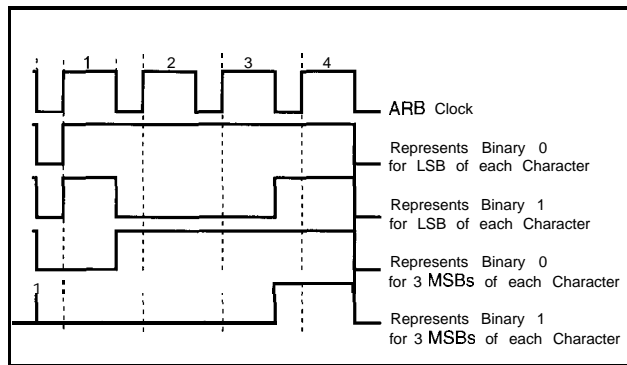


Figure 2—This took some thought. Notice the unique syncing patterns in phase 1.

First, trash the EXPLR2 demo system. This means, if you don't own a full-blown QNX license, go directly to jail and don't pass Go. Fortunately, I happen to have one.

There is some good news. The SLANG that comes with the demo is usable as is. Trashing the demo system implies adding some mechanical drives for now. No reason to develop an application without a full set of resources if you have them.

Once I modify the original EXPLR2 hardware layout and load QNX, I still can't bit-bang with the ARB.

I know you're thinking, "Why don't you just do this with C and a fast, embedded '486?" Well, if money's no object and you have the software on hand, go for it! If you don't think you'll ever have to maintain this system, go for it!

On the other hand, if you want to keep the cost of the project down, minimize maintenance time, and be creative in the process, shove a PIC in front of this operation.

Any PIC with the appropriate quantity of I/O pins can handle all the ARB-clocking, data-buffering, and data-conversion chores. These tasks aren't likely to change often-if ever.

This leaves the EXPLR2 with the ability to be totally programmed in SLANG. By sim-

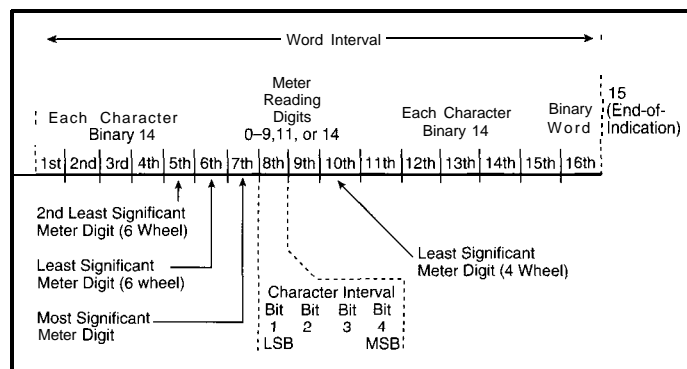


Figure 3—Data in the fifth and sixth character positions determines the actual register size.

ply employing a \$3 part, I get a workable and bumpless-update-capable Internet appliance.

The PIC interfaces directly to the ARB's clock/power and data lines. In that PICs are capable of operating with cycle times in the nanoseconds, it's second nature for these parts to produce the periods and duty cycles the ARB wants.

Once the data-collection cycle completes, the PIC can convert and store the ARB phases so they're ready to be processed immediately by the EXPLR2/SLANG Internet appliance.

The task is simple binary-to-ASCII conversion. The PIC looks after the ARB reading so no sorting or shifting is necessary at the SLANG end. The PIC's job is to deliver an ASCII reading identical to the external register of the ARB V.

The EXPLR2 serial port can be used to transfer the data between the ARB PIC and the EXPLR2. If a PIC with no internal UART is chosen, that entails extra PIC serial I/O code and thus more complexity.

Since ARB readings will more than likely be taken hours apart, there's plenty of time to transfer the readings via a couple parallel-port lines. This offload to the PIC leaves the EXPLR2 free to do what it does best—interface to the Web.

A Q U A W E B

Although the PIC saved the day on the ARB end, I'm still in deep water on the SLANG end. Unlike DOS-based systems, the typical QNX/SLANG programmer doesn't just go in and write directly to I/O devices.

This task is usually accomplished through specialized device drivers. But in this instance, writing a device driver is overkill.

I simply need to handshake with the PIC attached to the parallel port and insert the received ASCII meter reading into an HTML file. How can I do this if I can't do simple I/O to the parallel port?

The answer lies in something called privity. In the world of QNX, privity implies privilege.

To speak directly to the parallel port at address 0x378, my executable must possess a privity of 1. The privity executable isn't included with the EXPLR2 demo, but I had no trouble obtaining a copy from the folks at Cogent.

Having the executable and logging in as root, I set the privy for SLANG to 1 with the command:

```
privy 1 /usr/cogent/bin/slang
```

That one command puts us on the gravity-induced side of the waterfall, only three functions away from our goal. Reference the listings as I count them off.

The first task is to init the modem and dial the ISP (assuming it's not you). As you see in Listing 1, the dev_read and dev_writeSLANG commands make easy work of the modem setup.

Once all the data is assimilated, the write-data-to-file function builds an HTML file with the embedded ASCII reading. Before exiting, an ftp background function is spawned from within write-data-to-file, as shown in Listing 2.

The ftp program looks for a netrc file in the home directory of the user who started it. This file contains all the necessary information to automatically log on and transfer the HTML image.

The only thing left to do is define the data-collection intervals. SLANG's every

mnemonic enables the Internet appliance to read the PIC every hour and transmit a reading every 10 h.

That's it. Water on the Web.

WANNA WALK ON WATER?

Right now, designing Internet appliances is a big thing. If you decide to get your feet wet with the EXPLR2 demo system, you have 30 days to use the software licenses included with the board.

In other words, if you're serious about designing an Internet appliance using the tools offered in the EXPLR2 demo kit, be ready to purchase some software.

On the other hand, if you just need a worthy embedded platform for your Internet project, consider the EXPLR2. Internet appliances don't have to be complicated, just embedded. APC:IPC

Fred Eady has over 20 years' experience as a systems engineer. He has worked with computers and communication systems large and small, simple and complex. His forte is embedded-systems design and communications. Fred may be reached at fred@edtp.com.

SOURCES
Neptune ARB V
Schlumberger Water
Hwy. 229
Toll&see, AL
(334) 283-6555
Fax: (334) 283-7299

QNX OS
QNX
175 Terrence Matthews Cres.
Kanata, ON
Canada K2M 1 W8
(613) 591-0931
Fox: (613) 591-3579
www.qnx.com

SLANG
Cogent Real-Time Systems, Inc.
168 Queen St. S, Ste. 205
Mississauga, ON
Canada L5M 1 K8
(905) 8 12-9628
Fox: (510) 472-6958
info@cogent.ca
www.io.org/~cogent

EXPLR2
Intel Corp.
2200 Mission College Blvd.
Santa Clara, California 95052-8119
(408) 765.8080
Fax: (408) 765-9904
www.intel.com

IRS

- 4 19 Very Useful
- 420 Moderately Useful
- 42 1 Not Useful

Bridging the Gap to Better Service!

Hyper Peppy Robot Kit

- Changes course when comes in contact or hears a loud sound

Part No.	Description	Price
140863	Robot kit	Special \$19.95

MicroPlug™ PIC™ Prototyping System

- With this flexible design system, the perf area can be detached from the microcontroller area allowing changes without scrapping the board

Part No.	Description	Price
141170	Micro2 processor board for 28-pin DIPs	139272
139272	perf board for use with Micro2 processor board	141133
141133	parts kit for Micro2 board	141170

1 and 3 Axis Accelerometer Modules and Interface Card

- For motion sensing, vibration analysis and other acceleration applications

Part No.	Description	Price
141591	1 axis accelerometer	\$89.95
141575	3 axis accelerometer	159.95
141583	Digital interface card	199.95

EDWin NC CAD/CAE Software

- The first truly seamlessly integrated suite of software running in all Windows* formats...simulation, schematics and PCB design.

Part No.	Description	Price
141170	Micro2 processor brd.	\$17.95
139272	MicroPerf proto brd.	11.95
141188	Parts kit for Micro2	9.95

FOREFRONT POSTPlus Solution Card

- Three-digit display provides solution codes, and works even if the PC system or other PCSI code readers will not

Part No.	Description	Price
141356	POSTPlus	Special \$349.95

JAMECO

1355 Shoreway Road
Belmont, CA 94002-4100
FAX: 1-800-237-6948 (Domestic)
FAX: 415-592-2503 (International)
E-mail: info@jameco.com
http://www.jameco.com

Order Toll Free 24-Hours
7-Days a Week!

Call for your FREE catalog!

© 1997 Jameco 8/97

Mention V.I.P.# 8C7

Need An Adaptable Controller?



NEW!
PK2300 From \$179

No Problem!


Z-World's versatile PK2300 programmable controller adapts to your application. You can configure I/O as digital inputs, high current outputs, RS-485, or a resistance measurement input.

- 19 total I/O
- 11 user-configurable digital I/O lines
- DIN rail mounting
- Rugged ABS enclosure
- RS-232 and RS-485

Z-World offers cost-effective solutions for your control applications. Call today for a free catalog and more information on the new PK2300!

INNOVATION IN CONTROL TECHNOLOGY

Z-World, 1724 Picasso Avenue, Davis, CA 95616 USA
Telephone 916-757-3737 • FAX 916-753-5141
To place an order call 1-888-EMBEDUS (USA)
For immediate information, use our 24 hour AutoFax 916-753-0618



FEATURE ARTICLE

Gordon Dick

Test Drive a Precision Motion Controller

Want some quick and easy intelligent motion control? Gordon walks us through the step-by-step process he had in building an intelligent motion controller using National's LM628. Without much ado, his students could read I/O bits, process conditions, make decisions,....



First things first. What is a precision motion controller, or PMC, for short?

It's a chip that performs the intensive real-time computational tasks of implementing a high-performance digital motion-control system.

Feedback for such systems is usually from quadrature incremental optical encoders. Support circuitry and code are required to produce a functional intelligent motion-control system.

Many companies produce easy-to-use, intelligent motion-control cards that enable a system to be up and running quickly. These cards essentially stand alone, using a PC for communication and sometimes for power.

Motion-control code for subsequent execution is sent as a text file to the controller in its specified language.

This code is created in a text editor, or in more advanced situations, CAD drawings are translated into motion code. (I described building a complete, intelligent-card-based motion-control system in "Designing an Industrial-Grade XYZ Router Table," *INK 62*.)

The PMC I'll describe here—the National Semiconductor LM628—is the heart of a custom intelligent motion-control card (although to the host, it's just another I/O device). Commands and data are passed to the PMC over the host data bus.

Code for the PMC is no longer simply created in a text editor. Now, it must be assembled and linked as part of some executable micro code.

If you've built digital filters, you can appreciate how computationally intensive they are and how execution time grows.

The 8-MHz LM628, shown in Figure 1, can do the digital-filter calculations and all its other tasks once every 256 μ s.

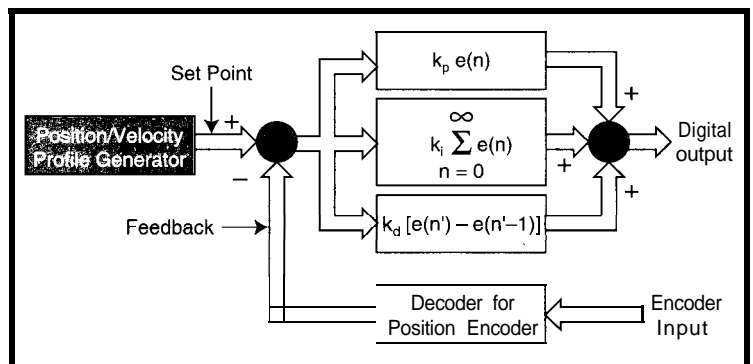
To make a functional intelligent motion-control system, you need a PMC (e.g., the LM628), host, DAC, DC servo motor, and incremental optical encoder. Figure 2 shows this interconnected collection of components.

IT'S TOOL TIME

So, let's get the necessary components and build an intelligent motion-control system! As it happened, I had a DC servo motor removed from a surplus printer, and it had an encoder already mounted on the back.

Although I had a servo amp removed from a surplus mag-tape unit, I didn't want to use it here. I'll eventually build about 12 of these systems, so I needed a servo amp for the prototype

Figure 1—A lot of number crunching is required for profile generation, the PID filter, and position decoding.



that could be built in small production quantities later.

A power op-amp makes a decent servo amp, but the price is usually scary. So, I settled on a modification of an audio amplifier using a National Semiconductor driver chip I'd been meaning to try for ages.

I didn't have a DAC removed from a surplus widget, so I used an off-the-shelf part. Getting parts in Edmonton is often a real challenge, so I wasn't all that picky about the DAC. It was 8 bit and in my hand!

Since these units are for a training course, I didn't need to build a power supply. It would be part of the test equipment at the bench. The schematic of the intelligent motion-control system prototype is presented in Figure 3.

Time to get out the wire-wrap tool and soldering iron and put these parts together. As usual, building proceeded quickly and I soon needed a "smoke test." The result of a capable summer student's wire wrapping and soldering is shown on the left side of Photo 1.

THE HARD PART

Have you ever had anything work the first time? Long ago, I discovered that the "power it up and see if it works" routine produced more smoke than a systematic checkout of each system block. Let's see how many problems I find as I examine the prototype block by block.

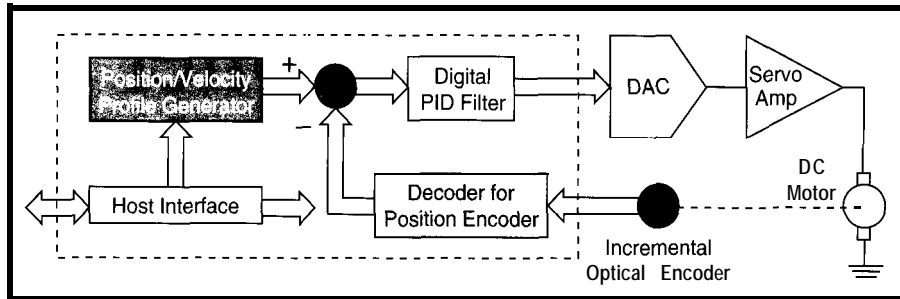


Figure 2—Add a DAC, a servo amp, a servo motor, and an optical encoder to the LM628, and you have a working system.

With no ICs in place other than the power-stage driver, does the servo amp work! Oddly enough, yes. Tweak the compensation capacitors to stop that tendency to oscillate, adjust the output offset voltage to zero, and trim the idle current in the power stage. Now, it's fine.

Next, I need to make the DAC feed a signal to the servo amp. After establishing logic levels at the digital inputs, I should have a related output voltage-but no. How did that active-low Latch Enable line get tied high? Fix that, adjust the reference voltage, and the DAC works fine, too!

It's time to install the LM628 and hook up the host. Many years ago at NAIT, we developed a microprocessor board as teaching tool. It's an 80C88-based system that's partly PCB and partly wire-wrapped. It's not state of the art, nor is it lightning fast, but it's an excellent vehicle for learning microprocessor basics.

Photo 1 shows the board with the LM628 connected to the host micro board. You can also see the servo motor and wiring to it.

Before anything else, communication between the host and the LM628 must be verified. I can partly test this by attempting to read the LM628 Status Byte, which can be read anytime.

Almost everything else on the LM628 has to be done by first checking to see if the device is busy or not. A read of the Status Byte shows it is a C4h—which is what it's supposed to be after a hardware reset. Therefore, the data-bus and control-line connections are correct.

National Semiconductor recommends a functionality test at this point that resets all the Interrupt flags and checks the Status Byte again. It should now read C0h or 80h, but instead it continued to read C4h.

Eventually, I discovered the LM628 active-low Reset was permanently wired high. After correcting that problem, the functionality test went as expected.

Now, I'm getting into the nitty gritty. Can I read position data from the encoder via the LM628 over the host data bus?

The testing up to this point was conducted without generating new code for the host. Part of the 80C88 micro board's firmware is a feature allowing data to be sent to or read from a particular I/O address. Until now, that's how the Status Byte was read.

However, the Busy Bit has to be checked, and multiple data bytes have to be read from the LM628. A short assembler program is required to proceed with testing to verify the encoder is working and being read correctly.

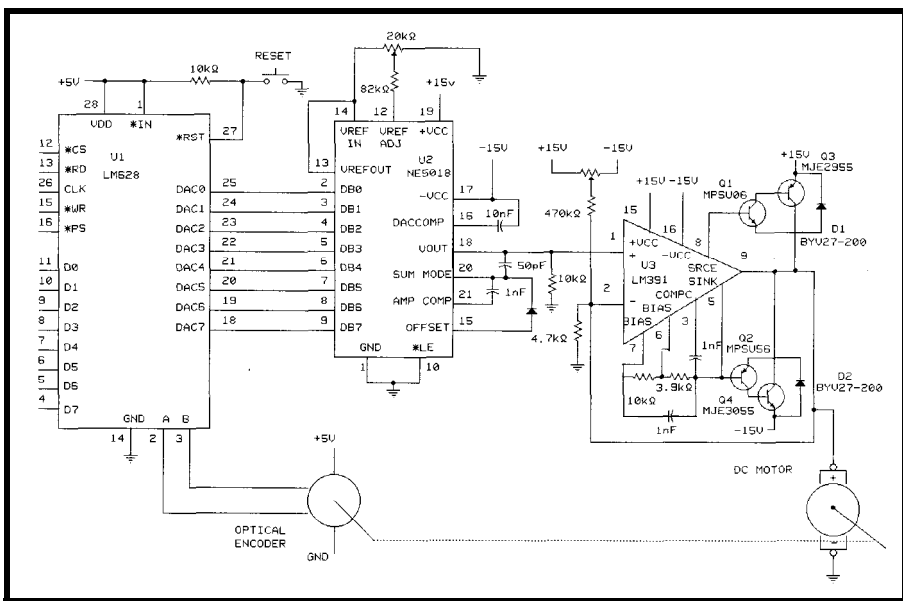


Figure 3—Aside from an amplifier a/ways wanting to be an oscillator, there's nothing critical here, except for the LM628.

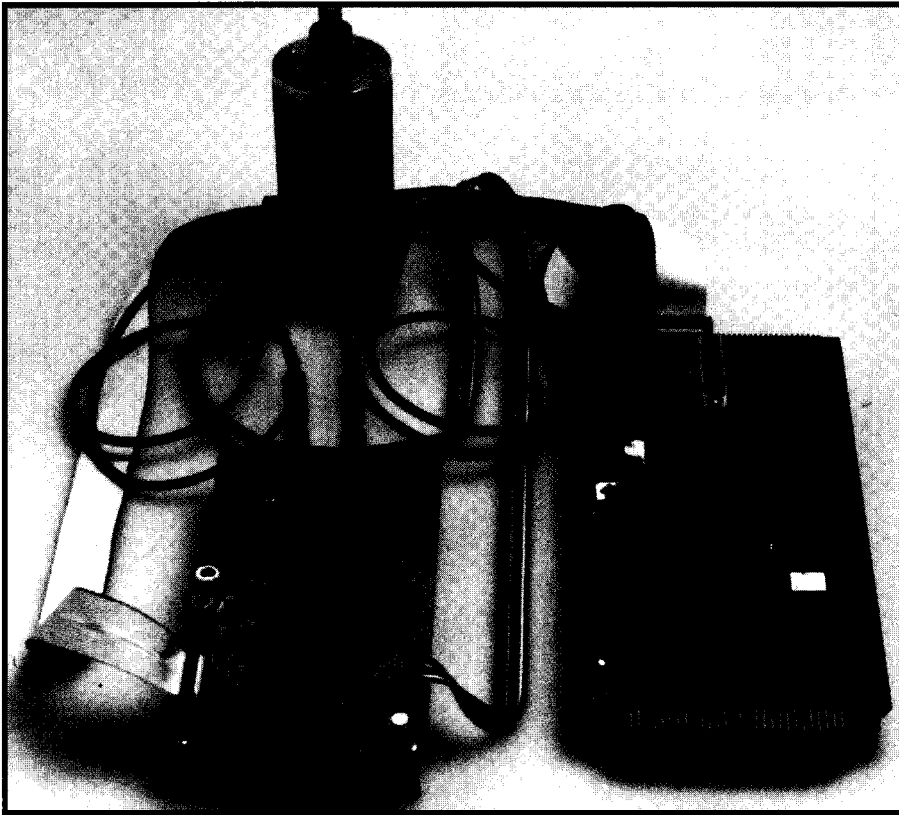


Photo I-k's almost a functional system. For communication to be up and running, add servo-amp power, 5 V for the host, and the serial connection from the host to a PC.

This is curious. The test program is getting data from the LM628, but it doesn't change when the shaft is rotated. Checking with a scope shows the encoder is functional. But, the pin on the LM628 for the Index pulse is open.

The encoder I'm using has no Index line, so the Index line on the LM628 should be tied high. When that's done, voila! I have 32-bit position numbers that track the shaft rotation.

To make the LM628 close the loop (i.e., start running the digital filter and produce an output proportional to position error, also called "servoing"), I need to assign values to a number of parameters. At the moment, most of the digital-filter coefficients are set to zero due to the hardware reset.

Establishing the digital filter parameters is a multistep process. The LFIL (load filter) command sequence begins with the LFIL command (1Eh) sent to the PMC command register. The filter control word follows and is sent to the PMC data register.

The first byte of the filter control word programs the derivative sampling interval, which is initially set the

same as the system sampling interval. So, this first byte is 00h.

The second byte indicates which filter coefficients follow. This process is explained well in the LM628 Programming Guide. For initial testing, it's wise not to get too fancy with the digital filter, so I used only a proportional gain, indicated by sending 08h as the second byte.

The expected data should follow. In this case, two bytes—00h and 28h—set the proportional gain to 40d, as you see in the first part of Listing 1.

I'm not quite finished with the filter. The data just sent to the LM628 for the filter is held in a buffer. No data is actually written to the digital filter until a UDF (update filter) command (04h) is sent to the PMC command register.

Before the LM628 will servo, it must have information about the upcoming move. This information is sent via the LTRJ (load trajectory) command (1Fh) to the PMC command register, followed by a number of words to the PMC data register.

The result is similar to what was done for the filter. The first word fol-

lowing LTRJ is the trajectory control word, indicating whether to execute a position or velocity move.

The second byte indicates which of the three trajectory parameters will be loaded. Since, for the moment, I don't want to move but only want to servo, the second byte is 00h. That value indicates no acceleration, velocity, or position data is coming.

Similar to LFIL, the data for LTRJ is held in buffers until the motion (STT) command (01h) is sent to the PMC command register. Listing 1 shows the code associated with LTRJ and STT.

The motor is still not connected to the servo amp, but the encoder is connected to the PMC. So, I'll execute the code in Listing 2 to send the necessary data to the PMC and connect a DVM to the servo-amp output terminals.

Rotating the motor shaft a small amount should produce a voltage at the servo-amp output terminals. Rotating the motor the other direction reverses the sign of the voltage measured. If this happens, the LM628 is running the digital filter and trying to servo. Good news!

At this point, if you connect the motor, you have a 50/50 chance of having a negative feedback system. If your luck is as bad as mine, then you have a 100% chance of being wrong, and arbitrarily connecting the motor will result in an unstable system that runs away.

To be sure the motor is connected correctly, try this simple phasing procedure. If you rotate the motor CW and the PMC produces a positive voltage at the servo-amp output terminals, connect the motor so the voltage on these terminals produces a CCW rotation.

This change makes the servo amp drive the motor in a direction that minimizes the position error. This phasing procedure is described in more detail in Chuck Raskin's book [1].

When you're confident about how to connect the motor, power down and connect it. Power up again, and run the code to servo. If things go right, trying to rotate the motor shaft now should be met with resistance.

Typically, you'd start off with low gains here until you think things are

right and then gradually increase the gain until you had a tight loop. Eventually, you want to incorporate some integral and derivative control as well. This may lead to some instability and require some tuning, but that's to be expected.

Getting the loop to servo is the last hard part. Once the negative feedback system is well-behaved, the rest is easy. Just sit at your terminal and generate code.

MAKING A MOVE

Some interesting applications are possible with an intelligent motion-control system. The host's ability to make decisions, read I/O bits representing process conditions, loop, and control I/O bits operating process elements enables you to produce some pretty sophisticated automation.

But, let's just do one simple position move, and then I'll point out a few of the other commands with interesting possibilities.

This example demonstrates the trapezoidal velocity move profile, so I intentionally chose a small value for acceleration. It's also worthwhile spending some time finding out the system's capabilities. Attainable speed is limited by the available motor voltage, and attainable acceleration is limited by how much current the servo amp can supply to the motor.

The arbitrarily imposed move conditions in this example are well within the system's capabilities. Let's go over the calculations for a move of 30 revs at a velocity of 1.5 rev/s using an acceleration of 1 rev/s/s.

In this case, the encoder is 1000 lines, and I'm using a 6-MHz LM628.

By watching for encoder pulse edges on the quadrature encoder signals, the PMC can improve resolution by four times.

The LM628 requires its move parameters in units of scaled counts per sample as follows. For velocity:

$$1.5 \frac{\text{rev}}{\text{s}} \times 4000 \frac{\text{count}}{\text{rev}} \times 341 \frac{\mu\text{s}}{\text{sample}} = 2.046 \frac{\text{count}}{\text{sample}}$$

Multiply by 65,536 to scale:

$$2.046 \frac{\text{count}}{\text{sample}} \times 65,536 = 134,086.656 \frac{\text{count}}{\text{sample}}$$

Then, truncate the fractional part and convert it to hexadecimal. Velocity to load is:

$$134086d \frac{\text{count}}{\text{sample}} \text{ or } 20BC6h \frac{\text{count}}{\text{sample}}$$

For acceleration:

$$1 \frac{\text{rev}}{\text{s}^2} \times 4000 \frac{\text{count}}{\text{rev}} \times \left(\frac{341 \mu\text{s}}{\text{sample}} \right)^2 = 0.000465 \frac{\text{rev}}{\text{sample}^2}$$

Multiply by 65,536 to scale.

$$0.000465 \frac{\text{rev}}{\text{sample}^2} \times 65,536 = 30.474 \frac{\text{rev}}{\text{sample}^2}$$

Then, truncate the fractional part and convert it to hexadecimal:

$$30 \frac{\text{rev}}{\text{sample}^2} = 1Eh \frac{\text{rev}}{\text{sample}^2}$$

For position:

$$30 \text{ rev} \times 4000 \frac{\text{count}}{\text{rev}} = 120,000 \text{ count}$$

Finally, convert it to hexadecimal, but don't scale this time:

$$120,000d \text{ count} = 1D4C0h \text{ count}$$

Load these move parameters into the PMC, and execute the move (see Listing 2). You should clearly see the motor shaft gradually accelerate up to speed, run at a constant speed, and gradually slow to a stop at the desired position.

A check of the actual position at the end of the move shows some error. An error of 10–20 counts is possible, depending on the Kp loaded into the filter and on the gain of the servo amp.

Introducing some integral gain (Ki) can reduce this small positioning error to zero, but it also destabilizes the system. Adding some derivative gain

Listing 1—Make the controller "control" by running the digital control algorithm.

```

PMC_CMD equ 60h ; PMC command register
PMC_DATA equ 61h ; PMC data register
LFIL equ 1Eh ; Load filter parameters opcode
UDF equ 04h ; Update filter parameters opcode
LTRJ equ 1Fh ; Load trajectory parameters opcode
STT equ 01h ; Start motion-control opcode

.model tiny
.code
.org 100h

Start:
; Reset LM628 before :ode is run the first time,
; The BUSY procedure s not shown here, but is included in the
; code available online.
; Load digital filter parameters.
  Mov al, LFIL          Load load filter parameters opcode
  Call BUSY            Wait until LM628 is ready
  Out PMC_CMD, al     ; Send opcode
  Mov ax, 0800h        ; Filter control word is 0008h, which sets
  Call BUSY            derivative sampling interval to 2048/fc1k
  Out PMC_DATA, al     and indicates that only Kp follows
  Xchg al, ah
  Out PMC_DATA, al :
  Mov ax, 2800h        ; Sets Kd to 40d
  Call BUSY
  Out PMC_DATA, al :
  Xchg al, ah :
  Out PMC_DATA, al :
  Mov al, UDF          Load update filter opcode
  Call BUSY            Wait until LM628 is ready
  Out PMC_CMD, al     ; Load new filter parameters
; Close servo loop.
  Mov al, LTRJ        Load load trajectory opcode
  Call BUSY            Wait until LM628 is ready
  Out PMC_CMD, al     ; Send opcode
  Mov al, 00h         Trajectory control word of 0000h
  Call BUSY            indicates to LM628 that no
  Out PMC_DATA, al : acceleration, velocity, or position
  Out PMC_DATA, al : data is to follow
  Mov al, STT         Run digital filter to close feedback loop
  Call BUSY
  Out PMC_CMD, al :
  Ret                 Return to monitor program
END Start

```

Finally, Standard RS-485 Network Software

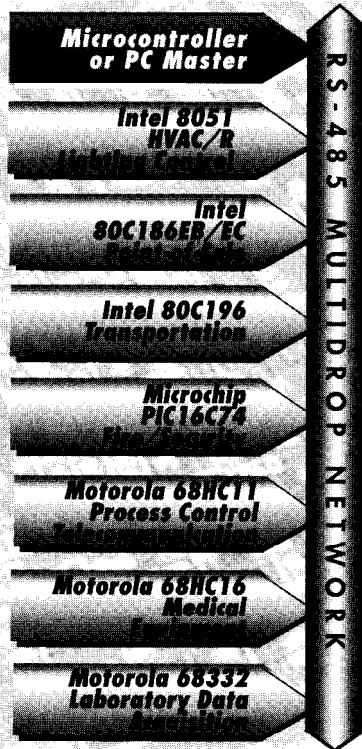
With Cimetrics' 9-Bit PLAN you can link together up to 250 of the most popular 8- and 16-bit microcontrollers (8051, 80C196, 80C186EB/EC, 68HC11, 68HC16, 68332, PIC16C74).

The 9-Bit μ LAN is:

- **Fast-** A high speed (62.5k baud) multidrop master/ slave RS-485 network
- ▶ **Flexible-** Compatible with your microcontrollers
- ▶ **Reliable** — Robust 16-bit CRC and sequence number error checking
- **Efficient-** Low microcontroller resource requirements (uses your chip's built-in serial port)
- ▶ **Friendly-** Simple-to-use C and assembly language software libraries, with demonstration programs
- **Complete-** Includes network software, network monitor, and RS-485 hardware
- **Standard-** The 9-Bit μ LAN is an asynchronous adaptation of IEEE 1118



55 Temple Place • Boston, MA 02111-1300 • Ph 617.350.7550 • Fx 617.350.7552



(Kd) helps stabilize the system, but I'll leave that for you to experiment with.

OTHER NEAT FEATURES

Being able to divide one motor revolution into 4000 parts and then move the motor shaft to a position accurate to a few of those 4000 parts is pretty impressive. But, the LM628 also offers additional sophistication, as far as move programming is concerned.

Until now, I've only used the PMC commands necessary to get the system working. The LM628 has 23 commands, and I'll briefly discuss a few of them here.

- **L P E I** (load position error for interrupt)-allows a level to be established beyond which an interrupt signal is sent to the host to announce an excessive error condition. Something may be stopping the motor from turning, for example, or the velocity of a move may be set higher than the motor can achieve. The host must handle the error condition.
- **L P E S** (load position error for stopping)-is similar to L P E I, except the PMC takes corrective action by stopping the motor
- **S B P A** (set breakpoint position absolute)-produces an interrupt when a specific absolute position is reached. The host can then initiate some activity at various points along a motion (e.g., turning a tool on or off).
- **S B P R** (set breakpoint position relative)-is similar to S B P A, but the position is measured relative
- **R D S I G S** (read signals register)-allows a handful of status-type information to be transferred to the host.

A PRODUCTION RUN

After it was clear that the prototype worked correctly, the next step was to make production quantities. For us at the NAIT, that means enough for 12 pairs of students plus a spare or two.

I passed the prototype and working schematics to the technical services group for PCB layout and production-prototype construction. After a couple of small corrections, we were ready for production.

#132

EMBEDDED BIOS™

BIOS Adaptation Kit

Listen to What our Customers Say

"We're impressed by the documentation and the readability of the code." - M. Ryan

"We are very pleased with the General Software BIOS and look forward to working with you to bring our product to market." - R. Levaro

Embedded BIOS is well-structured and documented, and technical support at General Software is excellent. - J. Toivanen

our'sion to buy BIOS from Software." - P. Fillon

the product for embedded PC designs. You were absolutely right." - Josse

The BIOS for Consumer Electronics

Why You Should Choose Embedded BIOS, Too

- BIOS, DOS, Flash Disk With One Low Royalty
- Instant Boot, Console Redirection, & Much More
- Expert Support with Guaranteed Response Time
- We Work Closely With Acer, AMO, Intel, & RadiSys to Deliver you a Proven, Tested, Feature-Packed BIOS
- Millions of Units Already Licensed

BIOS Adaptation Kit Includes:

- Complete Source Code
- Binary Configuration Program
- Quick Start + Over 600 Pages of Printed Documentation

For Details, Call 1-800-850-5755

GENERAL
GS

General Software, Inc.
320 108th Ave. N.E., Suite 400 • Bellevue, WA 98004
Tel: 206.454.5755 • Fax: 206.454.5744 • Sales: 800.850.5755
http://www.gensw.com/general • E-Mail: general@gensw.com

© 1997 General Software, Inc. General Software, the GS Logo, Embedded BIOS and Embedded DOS are trademarks of General Software. All rights reserved.

WRAPPING IT UP

At NAIT, we use this lab unit as an integral part of our Intelligent Motion Control course. The LM628 proved an excellent vehicle for exploring intelligent motion, particularly since the

students are also exposed to an intelligent motion card (a Galil DMC-620).

Giving students the opportunity to see how easy motion tasks can be implemented using an intelligent card and how significant the software task

becomes when you use a device like the LM628 helps them appreciate the bigger picture.

The LM628 was a good choice for this application for several reasons. Because it's a "mature" part, helpful application information was available. And, even though this chip is rather expensive (about \$47), National provided evaluation samples. So, I had the prototype up and running without spending any money.

Also, this chip is relatively easy to use. I've looked at others with far more intimidatingly thick manuals.

I'm in the process of examining other intelligent motion devices for use in the course. The next system I'll include is an intelligent stepper motor controller which communicates to a PC via the printer port. □

Gordon Dick is an instructor in electronics at the Northern Alberta Institute of Technology, Edmonton, AB, Canada. He has been involved in intelligent motion both as a consultant and an educator. The project presented here supported a training course on intelligent motion. You may reach Gordon at gordond@nait.ab.ca.

Listing 2—The first move requires a long series of commands and data to be sent. Subsequent moves can be made with fewer commands and data if velocity, acceleration, and other parameters are unchanged.

```
, The equates from Listing 1 are required.
.model tiny
.code
org 100h

Start:
; Make sure LM628 is Reset before code is run the first time.
; BUSY procedure is not repeated here but IS required.
; Load the digital filter parameters.
Mov al, LFIL ; load load filter parameters opcode
Call BUSY ; wait until LM628 is ready
Out PMC_CMD, al ; send opcode
Mov ax, 0800h ; filter control word is 0008h, which sets
Call BUSY ; derivative sampling interval to 2048/fclk
Out PMC_DATA, al ; and indicates that only Kp follows
Xchs al, ah
Out PMC_DATA, al ;
Mov ax, 2800h ; sets Kp to 40d
Call BUSY
Out PMC_DATA, al ;
Xchg al, ah
Out PMC_DATA, al ;
Mov al, UDF ; load update filter opcode
Call BUSY ; wait until LM628 is ready
Out PMC_CMD, al ; load new filter parameters
; Set up for relative position move of +120,000 counts.
Mov al, LTRJ ; load load trajectory op code
Call BUSY ; wait until LM628 is ready
Out PMC_CMD, al ; send opcode
Mov ax, 2B00h ; trajectory control word of 002Bh
Call BUSY
Out PMC_DATA, al ; indicates al | 3 move parameters are coming
Xchg al, ah ; next and that position will be relative
Out PMC_DATA, al
Mov al, 0h
Call BUSY
Out PMC_DATA, al ; send hi part of acceleration
Out PMC_DATA, al ; double word ie 0000h
Mov ax, 1E00h
Call BUSY
Out PMC_DATA, al ; send lo part of acceleration
Xchg al, ah
Out PMC_DATA, al ; double word ie 001Eh
Mov ax, 0200h
Call BUSY
Out PMCCDATA, al ; send hi part of velocity
Xchg al, ah
Out PMC_DATA, al ; double word ie 0002h
Mov ax, 0C60Bh
Call BUSY
Out PMC_DATA, al ; send lo part of velocity
Xchg al, ah
Out PMC_DATA, al ; double word ie 0BC6h
Mov ax, 0100h
Call BUSY
Out PMCCDATA, al ; send hi part of position
Xchg al, ah
Out PMC_DATA, al ; double word ie 0001h
Mov ax, 0C0D4h
Call BUSY
Out PMC_DATA, al ; send lo part of position
Xchg al, ah
Out PMC_DATA, al ; double word ie 04C0h
Mov al, STT ; load move parameters and start motion
Call BUSY
Out PMC_CMD, al
Ret ; return to Monitor program
END Start
```

REFERENCES

- [1] C. Raskin, *Designing With Motion Handbook II*, Technology 80, 1994.
National Semiconductor, *LM628/629 Precision Motion Controller*, Datasheet, 1994.
National Semiconductor, *LM628 Programming Guide*, App. Note AN-693, 1990.
National Semiconductor, *LM628/629 User Guide*, App. Note AN-706, 1990.

SOURCES

LM628
National Semiconductor
P.O. Box 58090
Santa Clara, CA 95052-8090
(408) 721-5000

I R S

422 Very Useful
423 Moderately Useful
424 Not Useful

70 MicroSeries

76 From the Bench

82 Silicon Update

Jan Axelson

Using Serial EEPROMs

Putting It All Together

Part
2
of
2

Building on last month's introduction, Jan tailors examples of how to use the serial EEPROMs for Microwire, SPI, and I²C. With a bit of Visual Basic code, you can access ports, use the programmer, and get inside the software.



Serial EEPROMs are popular devices for storing user settings, measurement data, and other changeable, nonvolatile information.

In a typical use, a microcontroller acts as a master that controls communications with the EEPROM. Many microcontrollers have built-in ports that are compatible with the EEPROMs' synchronous serial interfaces.

But, a controller isn't the only way to communicate with these devices.

A PC's parallel printer port is an inexpensive and flexible interface you can use for programming and reading serial EEPROMs. With a parallel-port interface, you only need to add some generic buffers and drivers and a cable.

And on the programming side, you can use any language that enables you to read and write to ports. Software can emulate all three of the popular synchronous interfaces, and the code can be easily modified to handle device variations.

In this installment, I describe the circuits and Visual Basic code for a parallel-port programmer for serial EEPROMs of the three interface types—Microwire, SPI, and X-introduced in Part 1.

I tested the programmer with a 4-Kb device of each type. With minimal

modifications, you can use the interfaces and program code to communicate with just about any serial EEPROM. You can also use the code and circuits as a starting point for talking to other chips that use similar interfaces.

ABOUT THE PROGRAMMER

One drawback of using the parallel port is that it's software intensive. If you use a microcontroller or expansion card with a built-in interface, the hardware handles most of the details of generating the clock and chip-select signals at appropriate times, dividing each byte to send into bits, and combining received bits back into bytes.

But, if you use the standard parallel port (or any generic I/O port), you have to do all this in software.

There's a variety of ways to connect the EEPROMs to the parallel port. Although many ports now include features for high-speed bidirectional communications, every PC's parallel port can emulate the original port's design with all bits under software control.

The signals are eight Data outputs (bits 0-7) at the port's base address, five Status inputs (bits 3-7) at base address + 1, and four Control outputs (bits 0-3) at base address + 2.

I tried several hardware configurations for the programmer. Figure 1 shows the interface I settled on because it was straightforward and usable on any PC's port.

Each EEPROM uses two or three of the Data outputs and one Status input. The Control port and two Status bits are unused. The eight ground returns in a standard 25-wire parallel cable all connect to signal ground in the EEPROM circuit.

Circuit construction and cable design aren't critical concerns. I used an ordinary 10 ribbon cable with the circuits on a solderless breadboard.

MICROWIRE AND SPI INTERFACES

The interfaces to Microwire and SPI EEPROMs are similar. Each line uses one of a 74LS244's buffer/drivers.

The DO and SO outputs aren't intended for driving long cables. They are guaranteed to sink at most a couple milliamps at 0.4 V, so I added a driver for each. But, in the other direction, again due to cable length, I used '244 buffers to add some hysteresis at the EEPROM's inputs.

The pullup at the Microwire's DO output is required only if you try to read the chip's Busy status after a programming operation completes. You don't need it if you read the status during a programming cycle or if you skip the Busy check entirely and just wait 10 ms to access the chip after programming it.

The choice of buffers and drivers isn't critical. A 74HCT244 or similar works well. And, if your cable is very short, you may get by without any added buffers or drivers at all.

The ORG, HOLD, and Write-Protect inputs are all tied inactive. If you want to control these in software, you can connect them to the unused Control-port outputs.

I²C INTERFACE

The I²C interface differs because it uses a single bidirectional data line (SDA). The EEPROM's SDA output is open drain, and the master's SDA output must be open drain or open collector as well, so either the master or the EEPROM can pull the SDA line low.

The I²C standard also specifies that SCL's output should be open drain to

enable multiple masters to take turns providing the clock signal. With a single master, you can use other output types.

One way to connect the SDA line to a PC's parallel port is to use a bit from the parallel port's Control port. On the original PC and most of its descendants, the Control bits are open collector with 4.7-k Ω pullups typical.

For faster switching on many of the newer ports, the Control outputs switch to push-pull type when the port is configured for a high-speed (EPP or ECP) mode. When emulating the original port, however, they revert to open drain.

But, a few ports don't have the open-collector/-drain outputs, so I didn't assume they'd be available. Instead, as with the other interfaces, I used a Data output and Status input.

SDA's input buffer is a 7407 open-collector driver with a 4.7-k Ω pullup. When the PC is writing

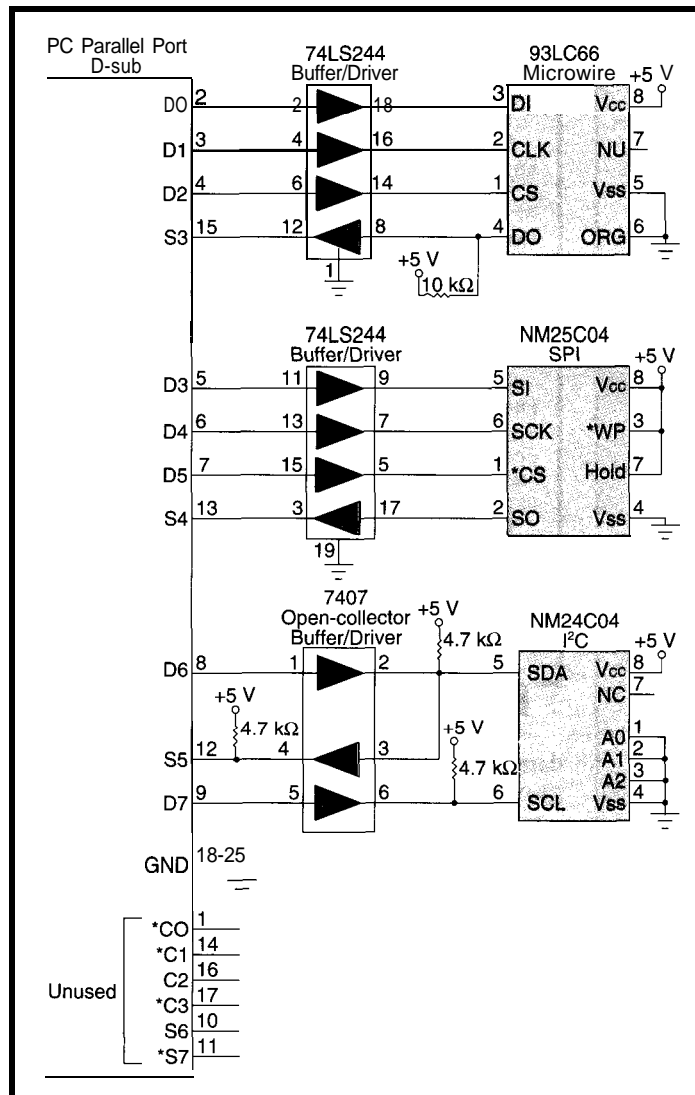
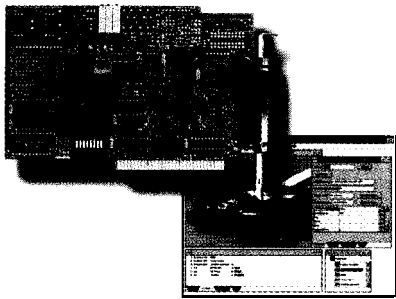


Figure 1—The PC's parallel port provides a simple interface for communicating with serial EEPROMs.



Lowest Cost Data Acquisition

ADAC's new Value-line has uncompromising design features and high quality components at prices below the low cost guys!

Just check out the specs:

Lowest Cost

5500MF
8 channels 12-bit A/D,
16 digital I/O, Counter/Timer

\$195

High Speed

5508LC
8 channels 12-bit A/D,
100KHz, DMA

\$245

Multi-Function DMA

5516DMA
16 channels 12-bit A/D,
DMA, 16 digital I/O

\$295

High Resolution

5500HR
16 channels 16-bit A/D,
DMA, 8 digital I/O

\$595

learn more:

voice **800-648-6589**
fax **617-938-6553**
web **www.adac.com**
email **info@adac.com**

ADAC

American Data Acquisition Corporation
70 Tower Office Par&, Woburn, MA 01801 USA

Listing 1—Each serial-EEPROM type uses a different protocol for sending and receiving bits. When using the PC's parallel port to communicate with the EEPROM, the software has to provide the clock transitions at appropriate times. These routines write bits to the EEPROMs.

```
' Bit numbers of output signals at parallel port's Data port.
'MW (Microwire):
Const DIN = 0
Const CLK = 1
Const CS = 2
'SPI:
Const SI = 3
Const SCK = 4
Const nCS = 5
'I2C (no hardware chip select):
Const SDAout = 6
Const SCL = 7

' Bit numbers of input signals at parallel port's Status port
'MW:
Const DOut = 3
'SPI:
Const SO = 4
'I2C:
Const SDAIn = 5

Private Sub I2CWriteBit(BitToWrite%)
' Write bit with SCL=0 and bring SCL high to latch bit into EEPROM
  DataToWrite = fncBitWrite(DataToWrite, SCL, 0)
  Out OutputPortAddress, DataToWrite
  DataToWrite = fncBitWrite(DataToWrite, SDAIn, BitToWrite)
  Out OutputPortAddress, DataToWrite
  DataToWrite = fncBitWrite(DataToWrite, SCL, 1)
  Out OutputPortAddress, DataToWrite
End Sub

Private Sub MWwriteBit(BitToWrite%)
' Write bit on CLK's falling edge
' and bring CLK high to latch data into EEPROM.
  DataToWrite = fncBitWrite(DataToWrite, DIN, BitToWrite)
  DataToWrite = fncBitWrite(DataToWrite, CLK, 0)
  Out OutputPortAddress, DataToWrite
  DataToWrite = fncBitWrite(DataToWrite, CLK, 1)
  Out OutputPortAddress, DataToWrite
End Sub

Private Sub SPIwriteBit(BitToWrite%)
' Write bit on SCK's rising edge
' and bring SCK low to latch data into EEPROM.
  DataToWrite = fncBitWrite(DataToWrite, SI, BitToWrite)
  DataToWrite = fncBitWrite(DataToWrite, SCK, 1)
  Out OutputPortAddress, DataToWrite
  DataToWrite = fncBitWrite(DataToWrite, SCK, 0)
  Out OutputPortAddress, DataToWrite
End Sub
```

data, addresses, or instructions to the EEPROM, SDA's output is off and SDA follows bit D6. During Read operations, D6 must be high to enable SDA to control S5.

You can use just about any LSTTL or HCTMOS buffer/drivers at S5 and D7 (SCL). I used the 7407s only because I was already using the chip and had the extra drivers. And of course, if you don't use open-collector or open-drain devices for these, you don't need the pullups.

If you use one of the parallel port's Control bits to communicate with SDA, be aware that bits 0, 1, and 3 in the parallel port's Control register read the inverse of the logic state at the connector. So, with them, remember to use inverting buffer/drivers or invert the bit in software.

USING THE PROGRAMMER

Photo 1 shows the user screen for the programmer application. I created the software with Visual Basic 4, and it

loads and runs in either the 16- or 32-bit edition under Windows 3.1 or Windows 95.

The software won't run under NT, which requires a kernel-mode driver for port accesses. If you have a driver for port I/O under NT, you can modify this program's routines.

A drop-down list box lets you select any of the three most common base addresses for parallel ports (i.e., 378h, 278h, 3BCh). You can use other addresses by adding them to the list box's code.

The application can program and read individual bytes or files. To program a byte, select the EEPROM type, enter the byte and an address in the text boxes, and click on the corresponding program command button.

After programming, the software reads the EEPROM's status, waiting for it to return a "not busy." The text box at the bottom of the window tells when the programming operation is completed or that the programming operation has timed out without receiving the expected response from the

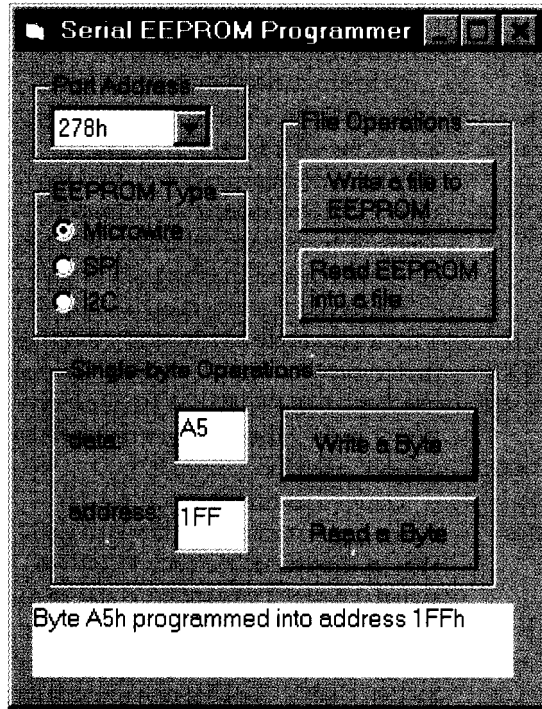


Photo 1—With this Visual Basic program and Figure 1's circuit, you can read and program Microwire, SPI, and I²C serial EEPROMs.

EEPROM. If the file to program is longer than 512 bytes, the software programs the first 512 bytes.

To read a byte from an EEPROM, select the EEPROM type, enter an

address in the text box, and click on the corresponding Read command button. The program reads the requested byte and displays it in the text box.

If it's an I²C interface and the software doesn't receive the expected Acknowledge signals, the read operation times out and displays a message. The Microwire and SPI interfaces have no time-out for read operations because the EEPROMs don't send Acknowledges.

To program a file's contents into an EEPROM or to write the contents of

Parts List Manager

For Engineers, Product Designers, Prototypers

Windows-based software helps you stay organized...And makes the job easier!

and manage multi-assembly parts lists for products in development...And after!

Keep track of:

- Part Specs & Dwg's
- Suppliers & Mfrs
- Boms and Kit Lists
- Product Costs
- Engineering Stock

Part No.	Description	Quantity	Unit Price	Total Price	Notes
126-02	Screen Ranged/Net 812.30.55	4138.00	2000	\$827,600.00	
126-03	Screen Ranged/Net 810.30.55	4781.00	2000	\$956,200.00	
126-04	Screen Ranged/Net 812.30.55	8240.00	2000	\$1,648,000.00	
126-06	Screen Ranged/Net 812.30.55	4290.00	2000	\$858,000.00	

Parts & Vendors™ Version 2.8

Version SE: \$99 + s/h Extended: \$299 + s/h

Call 800-280-5176

916-273-1985
fax 916-477-9106
P.O. Box 2270, Grass Valley, CA 95945
<http://www.trilogydesign.com>

Trilogy DESIGN

Requires 486, 10 meg min. ram. Win 3.x or Win95

THIS IS THE CAD FOR YOU!!!

"Easy to use, designed to help"

Now in Both Windows and DOS

CAD PAK Windows-\$199, DOS - \$159

Ideal for New Users, Hobbyists, & Small Businesses!
Provides everything for PCB Layout & Schematic Drawing

PROTEUS Starts at \$425

Most Powerful EDS System on the market!!!

- Schematic Capture * Circuit Simulation
- PCB Layout with Rip-Up & Retry Autorouting

R4 SYSTEMS INC.
1100 GORHAM ST. Suite 11B-332
NEWMARKET ONTARIO
CANADA L3Y 7V1
905 898-0665 FAX 905 898-0683
BBS 905 898-0508 (9600,8,N,1)

TR
FREE
WRITE
T

OUR DEMO OR CALL DAY

Download Demo - <http://www.r4systems.on.ca>
Internet email info@r4systems.on.ca

an EEPROM into a file, use `File Program` and `File Read`. Each brings up a common dialog box that lets you select a file to read from or write to.

A completed programming operation doesn't guarantee success. To verify, read the byte(s) back and compare with the original.

ACCESSING PORTS

The first challenge to accessing the parallel port in Visual Basic is that VB doesn't include BASIC's usual `Inp` and `Out` for accessing I/O ports.

A solution is to use an "Inpout" DLL that adds these routines to VB. The DLL reads and writes directly to the selected port.

The EEPROM programmer uses either of two DLLs, depending on if the program is running under the 16- or 32-bit edition of VB. As with all DLLs, the DLL itself must be on the system running the program and the program must declare the routines it calls.

The syntax for using the DLL's `Inp` and `Out` is the same as in QuickBasic:

```
ByteRead = Inp(PortAddress)
Out PortAddress, ByteToWrite
```

VB also allows this alternate syntax for `Out`:

```
Call Out (PortAddress,
ByteToWrite)
```

Another option for accessing ports under Windows 3.x or 95 is via a virtual device driver (VxD), which enables an application to block port accesses from unauthorized sources. A VxD has other benefits as well, such as the ability to respond more quickly and use system features like DMA.

However, both Windows 3.x and 95 allow direct port reads and writes as long as another driver hasn't blocked access to the port. If other applications don't need to use the port and if you don't need a VxD for other reasons, direct I/O is a quick and inexpensive solution.

INSIDE THE SOFTWARE

The program itself consists of many short routines. One set handles the user interface, including reading the

option buttons and text boxes and responding to button clicks.

Other routines handle tasks common to all three EEPROM types (e.g., extracting a bit from a byte and displaying time-out messages). And for each EEPROM type, a set of routines sends instructions, addresses, and data, and reads data in the required format.

I designed the program to work with an example 5 12-byte EEPROM of each type. With modifications, you can use it with EEPROMs of other capacities or make other changes required by a specific device.

Listing 1 shows the routines for writing and reading one bit with each type of EEPROM. For each EEPROM signal, I defined a constant equal to the signal's bit number at the parallel port. If you want to use different bit assignments, change the constants to match.

To keep the software as flexible and easy to understand as possible, I designed the code so you can change

individual bits in a byte without having to track the states of all the others.

A form variable (`DataOut`) holds the last value written to the Data port, and a `BitWrite` function sets or clears a selected bit in a byte. To toggle a bit at the Data port, the code first sets or clears the desired bit in `DataOut` and then writes the result to the port.

A challenge in getting this software working was that the serial links don't provide much in the way of feedback. The only way to know if a byte programmed successfully is to write the byte and read it back.

If it doesn't verify, there's no way to know if the problem was in the programming or read operation. A single missing or extra clock pulse or a mistake in an instruction or address means the intended operation won't complete.

PC sends an Acknowledge to let the master know when the EEPROM receives something. Even here, the master may read (logic low) Acks when

Listing 2—Each EEPROM type responds to a small instruction set. These routines send a Read instruction to the EEPROM, followed by the address to read. The EEPROM responds with the requested data. The PC and SPI instructions include address bit 8, while the Microwire instruction sends all nine bits following the instruction.

```
Private Sub I2CSendReadInstruction(A8%)
' Read instruction consists of device identifier (1010),
' two don't cares, address bit 8, and 1 (Read).
Call I2CIssueStartCondition
Call I2CWriteBit(1)
Call I2CWriteBit(0)
Call I2CWriteBit(1)
Call I2CWriteBit(0)
Call I2CWriteBit(0)
Call I2CWriteBit(0)
Call I2CWriteBit(0)
Call I2CWriteBit(A8)
Call I2CWriteBit(1)
Call I2CWaitForAck
End Sub

Private Sub MWSendReadInstruction()
'Sends Start bit (1) and Read instruction (1,0):
Call MWriteBit(1)
Call MWriteBit(1)
Call MWriteBit(0)
End Sub

Private Sub SPISendReadInstruction(A8%)
'Sends Read Instruction:
Call SPIWriteBit(0)
Call SPIWriteBit(0)
Call SPIWriteBit(0)
Call SPIWriteBit(0)
Call SPIWriteBit(A8)
Call SPIWriteBit(0)
Call SPIWriteBit(1)
Call SPIWriteBit(1)
End Sub
```

none have been sent (e.g., if the circuits aren't powered up). So only a successful verify, not the lack of an error message, indicates a success.

Fortunately, with all three interfaces, you can toggle the clock as slowly as you want. To troubleshoot, single-step through the routines and verify that each signal behaves correctly each time.

Listing 2 shows routines for writing Read instructions to each EEPROM type. Again, nothing is automatic. Software provides all the clock transitions and writes each bit of the instructions and data at appropriate times.

ENHANCEMENTS

Although the program is functional as it stands, chances are that you'll want to make changes and enhancements, such as the ability to use other EEPROM sizes or the addition of instructions such as `Erase All`.

Other enhancements might include saving program settings such as default EEPROM types and file directories as well as more robust error checking.

Jan Axelson is the author of Parallel Port Complete and The Microcontroller Idea Book. You may reach her by E-mail at jaxelson@lvr.com or via her Web site at www.lvr.com.

SOFTWARE

The complete program code is available at www.lvr.com and on the Circuit Cellar Web site.

SOURCES

68HC11
Motorola
MCU Information Line
P.O. Box 13026
Austin, TX 7871 1-3026
(512) 328-2268
Fax: (512) 891-4465
www.mcu.motps.com/mc.html

8xC528
Philips Semiconductor
811 E. Arques Ave.
Sunnyvale, CA 94088-3409
(408) 991-5207
Fax: (408) 991-3773
www.semiconductors.philips.com

Serial EEPROMs
Digi-Key Corp.
701 Brooks Ave. S
Thief Falls, MN 56701-0677
(218) 681-6674
Fax: (218) 681-3380

93LC66, 93C76, 24LC04
Microchip Technology, Inc.
2355 W. Chandler Blvd.
Chandler, AZ 85224-6199
(602) 786-7200
Fax: (602) 786-7277
www.microchip2.com/appnotes/appnotes.htm

COP888, NM25C04, NM24C04,
NM24C03
National Semiconductor
P.O. Box 58090
Santa Clara, CA 95052-8090
(408) 721-5000
Fax: (408) 739-9803
www.national.com/design

IRS

425 Very Useful
426 Moderately Useful
427 Not Useful

LOGICAL

*Versatility in device programmers DOS, Windows 3.1
Windows 95 Windows NT Compatible*



Starting at \$495.00

Start programming devices today with the lowest cost and highest per-

TSOP, QPF, PLCC, DIP.... programming heads. Evaluate a unit today with 100% satisfaction guaranteed or **YOUR MONEY BACK!**
(no penalties or restocking fees if unit is returned).

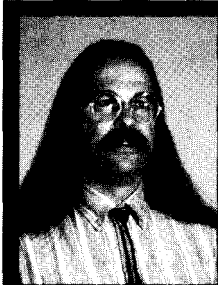
**Gang any Device, in any Socket Type
with PC based or Stand Alone unit**

Call Today in USA **800-331-7766**
Fax: 303-733-6868 or Visit our Home Page:

www.logicaldevices.com

It Can't Be A Robot

Part 3: It's Blind as a Bat



Knowing where you're at is critical, even for

robots. To avoid those fender-bender blues, Jeff coordinates the position transducers' feedback and motor driver using PicStic as the master processor.

FROM THE BENCH

Jeff Bachiochi



You may recall a project I did a few years ago using Polaroid ultrasonic transducers mounted on the rear of my motorcycle ("Probing the Dark Side," *INK* 50).

Five transducers kept watch for vehicular movement to my rear, corners, and sides. A display of colored LEDs indicated where the transducers saw objects. I used the full range of transducers.

Receiving long-distance (> 10') echoes is based on two factors. First, the objects reflecting the ultrasonic bursts must be of sufficient size to create a sizable echo.

As well, the voltage used to fire the transducers must be sufficiently high. A lot of energy is required to produce a powerful signal so there's enough of that signal to be reflected and an echo heard.

Signal strength is reduced by the square of the distance. And, the bit of signal that is reflected is also reduced by the square of the distance for the return trip.

Large robots might need to know about objects 30' or more away, but the little guy I've been working on won't have much use for that kind of information. Instead, his universe consists of knowing about only those things located within a few feet.

Since the distances will be considerably smaller than those associated with the Polaroid system I developed for my bike, I may be able to get away with a much less sophisticated device. Let's see what else is available for transducers.

My first experience with ultrasonics was from Heathkit. Behind the camouflage of a classic hardcover book was one of the first motion sensors I remember seeing.

The transducer and receiver were aimed through the false binding. A disturbance in what the circuitry saw as the normal echo pattern tripped a relay and could be used to perform any alarm-type function. Real cloak and dagger for a time when no one even thought of locking their front door.

The ultrasonic transducers in that kit were much different than my half-dollar-sized Polaroid ones. I located some similar units in the Mouser catalog. (Digi-Key has discontinued their transducers.)

REACH OUT AND DON'T TOUCH

The ultimate distance sensor would cover an area exactly the width of the object (i.e., the robot). Perfect coverage would show distance to any object that falls directly in the path of-motion.

It's difficult for a single sensor to know, however, where in its field of view the object resides (see Figure 1a).

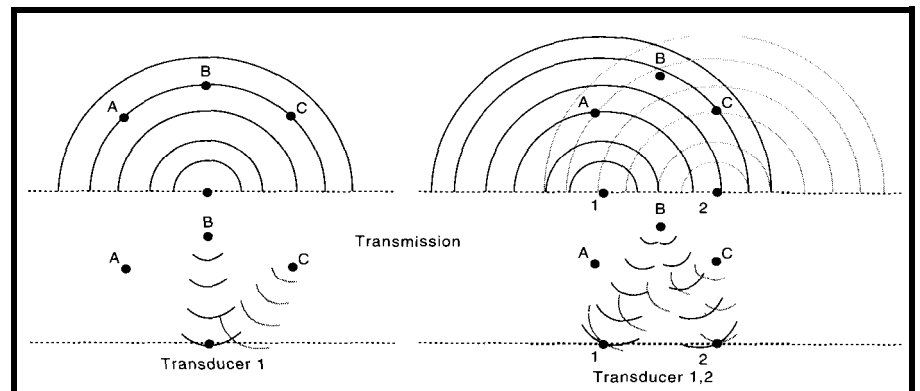


Figure 1a—By measuring the distance from the source to an object and the echo back, position is indeterminate using a single transducer. b-A much better view is obtained using two transducers.

Listing 1-A call to `_UL_TRA` produces 10 cycles of 40-kHz ultrasonic transmission. A 100- μ s loop is performed 255 times. This loop checks the four ultrasonic receivers and saves the loop count when an echo is first heard. These counts are returned and are used to calculate the distance.

```

ASM
_ULTRA  clrf    _B0
        clrf    _B1
        clrf    _B2
        clrf    _B3
        movlw   _CYC
        movwf   _B5
_U0     bsf     PORTB,5
        movlw   3
        movwf   _B4
_U1     decfsz  _B4
        got0    _U1
        nop
        nop
        bcf     PORTB,5
        movlw   2
        movwf   _B4
_U2     decfsz  _B4
        got0    _U2
        nop
        nop
        decfsz  _B5
        got0    _U0
_CNT    clrf    _B5                ; [1]
        movlw   21                 ; [2]
        movwf   _B4                ; 131
_U3     decfsz  _B4                ; [4] [67/67]
        got0    _C0                ; [5/6]
        nop                        ; [69]
        incf    _B5                ; [70]
        movlw   0FFh              ; [71]
        subwf   _B5,W              ; [72]
        btfs   STATUS,2            ; [73]
        got0    _TST0              ; [74/75]
        got0    done
_TST0   movf    PORTB, W           ; [1]
        nlowf   _B6                ; [2]
        btfs   _B6,0              ; [3] [3/4]
        got0    _TST00             ; [4/5]
        movf    _B0               ; 151
        btfs   STATUS,2            ; [6] [6/7]
        got0    _TST01             ; [7/8]
        movf    _B5,W             ; [8]
        movwf   _B0               ; [9]
        got0    _TST1              ; [10/11]
_TST00  nop                        ; [6]
        nop                        ; [7]
        nop                        ; [8]
_TST01  nop                        ; [9]
        nop                        ; [10]
        nop                        ; 1111
_TST1   btfs   _B6,1              ; [12] [12/13]
        got0    _TST10             ; [13/14]
        movf    -81                ; [14]
        btfs   STATUS,2            ; 1151 [15/16]
        got0    _TST11             ; [16/17]
        movf    _B5,W             ; [17]
        movwf   _B1               ; 1181
        got0    _TST2              ; [19/20]
_TST10  nop                        ; [15]
        nop                        ; [16]
        nop                        ; 1171
_TST11  nop                        ; [18]
        nop                        ; [19]
        nop                        ; [20]
_TST2   btfs   _B6,2              ; 1211 [21/22]
        got0    _TST20             ; [22/23]
        movf    _B2               ; 1231
        btfs   STATUS,2            ; 1241 [24/25]
        got0    _TST21             ; [25/26]
        movf    _B5,W             ; 1261
        movwf   _B2               ; [27]
        got0    _TST3              ; [28/29]
_TST20  nop                        ; [24]

```

(continued)

Although ultrasonic sensors are more akin to our ears than our eyes, when used in pairs, object distance can be triangulated similar to the way our brains estimate distance. Relevant position is determined by the distance difference to the same (assumed) target, as depicted in Figure 1b.

Similar positioning information can be acquired via a single sensor if it's moved (either by turning the robot or the sensor). Since fewer sensors mean not only less expense but also a much simpler system, I'll try this approach.

Let's begin with one sensor for each direction (front, rear, left, and right). In this fashion, the front and rear sensors can point out impending doom while moving forward or backward.

The side sensors can help navigate a safe distance from a wall by keeping the motion parallel to it. If motion is not kept parallel, the robot may get stuck by grazing the wall without the forward sensor seeing it.

HELLO...HELLO

In this situation, we're dealing with distances relative to the size of the robot, not the size of objects across the room. Since I've been using the PicStic micro thus far with the robotic platform, continuing to use the same inexpensive brain will keep this multi-processor system simple.

The PicStic can easily produce a 40-kHz transmission burst via a BASIC command and directly drive the transmitting transducer. However, I also want the PicStic to count the time it takes any echo to return to its receiver.

Since sound travels about 12.5 in./ms, I need to count in increments of less than a millisecond to get resolution down to about an inch. Counting in tics of 100 μ s, I get a resolution of **1.25"** per tic round trip (out and back) or 0.625" per tic (out to the object).

An 8-bit register can hold 255 tics, which is equivalent to well over 10'—much farther than will be necessary. Therefore, the maximum time spent sending out a transmission burst and testing at each tic for an echo will be about 30 ms (maximum tic count of 255).

\$200 4A C Compiler?

You heard right. A quality C compiler designed for the 8051 microcontroller family, just \$200, including the Intel compatible assembler and linker. A great companion to our fine Single Board Computers, like those below. CALL NOW!

552SBC

- 80C552** a '5 1 Compatible Micro
- 40 Bits of Digital I/O
- 8 Channels of **10** Bit A/D
- 3 Serial Ports (**RS-232** or **422/485**)
- 2 Pulse Width Modulation Outputs
- 6 Capture/Compare Inputs
- 1 Real Time Clock
- 64K bytes Static RAM
- 1** + UVPRM Socket
- 5** 12 bytes of Serial EEPROM
- 1 Watchdog
- 1 Power Fail Interrupt
- 1 On-Board Power Regulation

Priced at just \$299 in single quantities. Call about our 552SBC C Development Kit, just \$448.

99 MHz 8051!

Our popular 8031SBC can now be shipped with Dallas Semi's hyperactive **DS80C320**, an 8051 on steroids. Averaging 3x faster than the standard 51, your project can really scream! **Call or email for pricing and brochures today!**

Other versions of the 8031SBC have processors with on-chip capture registers, EEPROM, IIC, A/D and more. Call or email for a list!

8031SBC as low as \$49

Call for your custom product needs. Quick Response.

HTE HiTech Equipment Corp.
9672 Via Excelencia
San Diego, CA 92126
(Fax: (619) 530-1458)

Since 1983

(619) 566-1892



Internet e-mail: info@hte.com
World Wide Web: www.hte.com

Listing 1—continued

```

nop                                ; [25]
nop                                ; [26]
_TST21 nop                           ; 1271
nop                                ; C281
nop                                ; [29]
_TST3  btfsc  _B6,3                  ; [30] [30/31]
goto  _TST30                        ; [31/32]
movf  _B3                             ; [32]
btfss STATUS,2                      ; [33] [33/34]
got0  _TST31                         ; [34/35]
movf  _B5,W                          ; [35]
movwf _B3                             ; 1361
got0  _CNT1                          ; [37/38]
_TST30 nop                           ; [33]
nop                                ; 1341
_TST31 nop                           ; [35]
nop                                ; [36]
nop                                ; [37]
_CNT1  movlw  18                     ; 1381
movwf _B4                             ; c391
_C1    decfsz _B4                    ; [40]
goto  _c0                             ; [41] [95/96]
nop                                ; [42/43]
nop                                ; 1971
nop                                ; 1981
goto  _TST0                          ; [99/100]

endasm

```

BASIC is a bit slow for counting in the microsecond range. Luckily, it's easy to use assembly language with the PicStic. In fact, it can reside in the BASIC listing, which keeps all the code in a single file and is great for keeping revisions straight.

The receive transducer by itself produces small (millivolt) voltage swings even at only short ranges. It needs to be highly amplified to produce a usable signal.

Rather than build this special analog front end, I decided to cheat by using circuitry you may already be familiar with.

What frequency comes to mind when I mention the words "IR transmission"?

It'd have to be 40 kHz, right?

I took the Sharp GP1U5 IR receiver and replaced the IR sensor with an ultrasonic transducer and got a high-gain amplifier and 40-kHz demodulator in a COB (chip onboard) circuit giving a TTL output. The circuit diagram in Figure 2 gives a more complete picture.

The metal shield surrounding the device can be easily removed. It just snaps on and off. I mounted the transducer right to the little PCB inside, as you see in Photo 1.

Only the code for the transmission and reception is done in assembly. It is called directly from the BASIC code.

This setup lets me use BASIC for the main program loop (including communications). I didn't have to get bogged down with a total assembly-language program because speed is not required here, except for the transmission and tic counting.

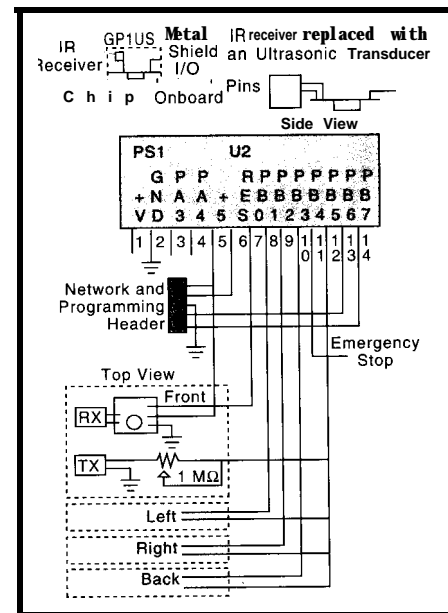


Figure 2—The PicStic micro becomes the network master and takes care of transmitting 400-kHz ultrasonic bursts while also listening for echoes on pins 1 through 14.

SAVE

53%

Off the

Cover

Price

\$21.95

U.S. Addresses

\$31.95 Canada & Mexico

\$49.95 all other

2-Year \$39 U.S. Addresses

\$55 Canada & Mexico \$85 all other

CIRCUIT CELLAR

(860) 875-2199

Fax (860) 871-0411

www.circuitcellar.com

Listing 1 shows the assembly-language routine called from BASIC.

WORKING TOGETHER

How can this measurement system be used with the motor-control system introduced in *INK 83*?

Since both systems are designed for network use, this measurement system can become the master processor and provide commands to the slave motor-drive processor.

Of course, there is a small matter of task. Without even the simplest of tasks, the platform will just sit there and do nothing.

To get the ball (er, robot) rolling, I chose the task of roaming an area and avoiding any obstacles it comes across. When the robot finds an obstacle, it must avoid collision by keeping its distance and maneuvering around it.

This task is broken down into two levels—moving and collision avoidance.

Listing 2—The BASIC portion of *PicStic's* program takes care of network communication and responds to ultrasonic range input from four discrete receivers positioned to the front, rear, left, and right of the robot.

```
Symbol S0 = 7          ' Serial Output Channel
Symbol SIN = 6         ' Serial Input Channel
Symbol CYC = 10        ' number of 40-kHz cycles to send
Symbol SAFMAX = 12
Symbol SAFMIN = 10
Symbol FWD = B0
Symbol BWD = B3
Symbol RGT = B2
Symbol LFT = B1

Start: peek $81,B0
      BO = BO & $7F
      poke $81,B0
      output 5
      output 4
      high 4
      pause 1000
Lev_1: Serout S0,N9600,("MF0",13,10)
Lev_1a: CALL ULTRA
      If FWD<SAFMAX or RGT<SAFMAX or BWD<SAFMAX or LFT<SAFMAX
      then E_STOP
      goto Lev_1a
E-STOP: iow 4
      Serin SIN,N9600,("MF0")
      high 4
Lev_2: CALL ULTRA
      If FWD<SAFMAX then Lev_2f
      If RGT<SAFMAX then Lev_2r
      If LFT<SAFMAX then Lev_2l
      goto Lev_1
Lev_2r: if RGT<SAFMIN then lev 2rl
Lev_2rr:gosub RIGHT
      gosub FORWARD
      goto Lev_2
Lev_2rl:gosub LEFT
      gosub FORWARD
      goto Lev_2
Lev_2l: if LFT<SAFMIN then lev_2rr
      goto Lev_2rl
Lev_2f: If RGT<SAFMAX then Lev_2fr
      If LFT<SAFMAX then Lev_2fl
      gosub RIGHT
      goto Lev_2
Lev_2fr:gosub LEFT
      goto Lev_2
Lev_2fl:gosub RIGHT
      goto Lev_2
RIGHT: Serout S0,N9600,("MR5",13,10)
      Serin SIN,N9600,("MR5")
      return
FORWARD:Serout S0,N9600,("MF5",13,10)
      Serin SIN,N9600,("MF5")
      return
LEFT: Serout S0,N9600,("ML5",13,10)
      Serin SIN,N9600,("ML5")
      return
```

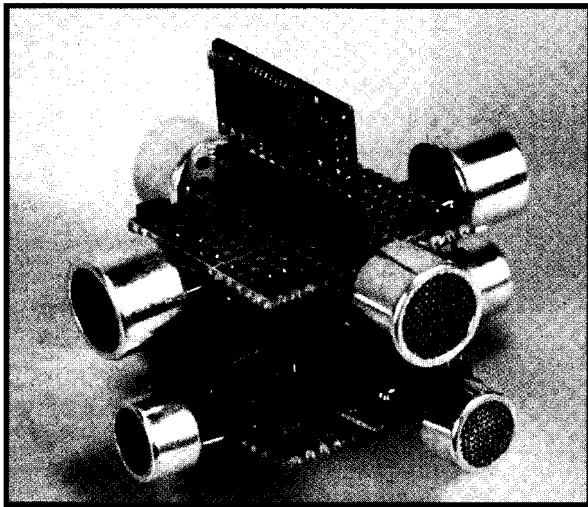


Photo 1--The top ultrasonic transmitters are all pulsed together. Echoes are caught by the lower receivers and the time difference becomes a function of echo distance travelled.

The first level-moving-is simple. The robot just turns on both platform treads and moves forward until told to do otherwise.

If you remember back to the motor-controller discussion (INK 83), the four commands were Fx, Bx, Lx, and Rx. That's forward, backward, left, and right, with x equaling the number of decoder counts to perform (0-255, where 0 has no count limit). The process continues until interrupted by an emergency stop input to the processor.

The master PicStic watches the sensors for an object to come within an unsafe distance. It immediately outputs an emergency stop to the slave PicStic, and the forward movement is halted. Now, the master processor drops into level-two mode-collision avoidance.

In level-two mode, movement is based on the sensor readings. If the forward sensor and/or either of the side sensors shows an obstacle at an unsafe distance, a small turn is implemented away from the obstacle until the forward sensor is clear.

If only a side sensor indicates an unsafe condition, small changes in direction are added to the forward motion in an attempt to maintain a safe yet parallel condition with the object. The main loop of Listing 2 has the particulars.

ON ITS OWN

The robotic platform is now free of the RF link demonstrated in Part 2. The next phase will be to come up with some more complex tasks for the

robotic platform. But, that's a task I leave for you.

Meanwhile, I'll let this little guy roam the office so I can further refine my simple and most likely imperfect behavioral assumptions. □

Jeff Bachiochi (pronounced "BAH-key-AH-key") is an electrical engineer on Circuit Cellar INK's engineering staff. His background includes product design and manufacturing. He may be reached at jeff.bachiochi@circuitcellar.com.

SOURCES

Ultrasonic transducers
Mouser Electronics
11433 Woodside Ave.
Santee, CA 92071
(619) 449-2222
Fax: (619) 449-6041

GPIO5 IR receiver/demodulator
Sharp Electronics Corp.
Microelectronics Group
5700 NW Pacific Rim Blvd., Ste. 20
Camas, WA 98607
(360) 834-2500
Fax: (360) 834-8903

PicStic 1
Micromint, Inc.
4 Park St.
Vernon, CT 06066
(860) 871-6170
Fax: (860) 872-2204
www.micromint.com

428 Very Useful
429 Moderately Useful
430 Not Useful

COMM-DRV/VxD	COMM-DRV/Lib
Built-in Multidrop protocol.	Supports Windows NT, 95, 3.x, and MS-DOS with same API.
Built-in 9-Bit Protocol.	Supports Visual C/C++, Borland, etc.
Supports all tools that can call DLLs.	Supports all tools that can call DLLs.
Bauds up to 460k.	Any # of ports.
Supports Windows 95 & Windows 3.x.	
Any # of ports.	

WCSC 4425 Kingwood Dr. Suite 2
Kingwood, TX 77339

Tel: (800) 966-4832 or (281) 498-4832 Fax: (281) 568-333
Visa/Mastercard/American Express/Discover/Wire Transfer/Approved P.

#137

COMPONENT-SIZED DAS

We put all this DAS capability in a tiny 28-pin package, and it still talks in simple English.

Power
+5 V-only or 9-16 V

Communication
0-57.6 kbps
Direct RS-232* or Networked RS-485*

Special Functions
Parallel printer output
PWM output
Dallas IButton S/N
Frequency input
Event counter input

Parallel I/O
8 bits bidirectional
Keypad scanning
Direct LCD control
High-current outputs

Actual size
1.5" x 0.875"

***External options in Answer MAN Jr.**

WWW.MICROMINT.COM
MICROMINT, INC.
1-800-635-3355 • (860) 871-6170

#119

Emulators for 8051/52, 251 PIC, 196 and PowerPC


In-Circuit Emulators Support

PowerPC series,
251 series,
80C186/188 series,
68302 series,
68307,

ColdFire series,
196 series,
Am186EX series,
68306,
PIC 16F series,

Real-time In-Circuit Development

- Windows-based Source Level Debugger
- Software Performance Analysis
- Trace and Trigger on-the-fly



More Information
Voice 408-955-0225
Fax 408-955-9705
E-mail misc-sale@microtek.com
<http://www.microtek.com/tw/mice>

MICROTEK

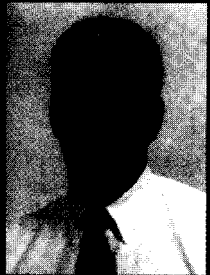
#138

Circuit Cellar INK® Issue 85 August 1997 81

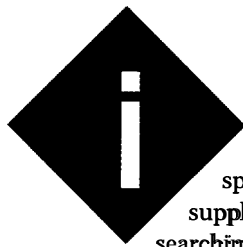
SILICON UPDATE

Tom Cantrell

Serial Flash Busts Bit Barrier



Just when you think you know how parts differ, they mutate. Or, so it seems with Nexcom's NX25Fxxx chip. It's a single-transistor EEPROM cell with block and chip erasure and SPI serial interface.



In this era of specialization, IC suppliers are constantly searching for new product

prospects. Once an idea is narrowed down to a specific class of products, it's helpful to make a table that cross-references key features with competitors' offerings, and then look for a hole to fill.

Of course, many times, this mechanistic approach spits out losers. In other words, don't bother trying to sell your boss on your bright idea for a x3 RAM chip.

Nevertheless, it's a good way to spin lots of ideas and encourages thinking outside the box. Hmm... 3 does divide into 9, 18, and 36, which are commonly used memory widths. You just never know whether

people will salute an idea until you run it up the flagpole.

Case in point is the ever-more-popular nonvolatile memory, which comes in a variety of forms including flash, EEPROM, FRAM, and battery- (or capacitor-) backed SRAM. Heck, for all I know, someone somewhere is still using bubble memory.

Anyway, a list of the key features starts with the most important ones (e.g., speed, power, density, and price) and proceeds to more detailed requirements such as byte versus block organization, write speed and endurance, interface, packaging, and so on.

Cross-referencing with all the available parts, one missing link leaps off the page. Yes, there are plenty of low-density (e.g., less than 64 Kb) chips with serial interfaces. Yes, there are plenty of high-density (e.g., greater than 1 Mb) chips with parallel interfaces. But, you guessed it, there aren't any high-density chips with serial interfaces.

Voilà, instant new product idea. Does it make sense? Nexcom Technology thinks so and is willing to ante up their NX25Fxxx chips to prove it. Read on, and you be the judge of whether they've got a winning hand.

MORE FLASH, LESS EEPROM

Since Nexcom's NexFlash chips are based on a single-transistor EEPROM cell, I'm not sure the traditional circuit-oriented nomenclature has much meaning anymore.

In my own mind, if it's byte or word erasable, it's an EEPROM. If it's block or chip erasable, it's flash.

To muddy the waters even further, Nexcom throws in a couple chunks of SRAM. The only way to figure out what the chip is is to see what it does. Let's take a look.

As shown in Figure 1, the Nexcom part starts with the already men-

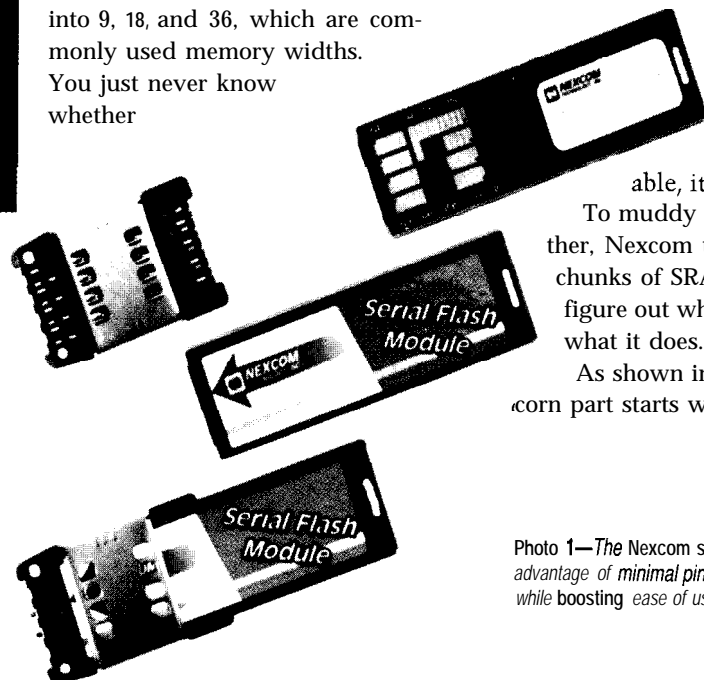


Photo 1—The Nexcom serial flash module takes advantage of minimal pin count to lower size and cost while boosting ease of use.

HR1 HR0 Pin Configuration		
0	0	Hold Input
0	1	No Connect
1	0	Ready/Busy Output (Open Drain)
1	1	Ready/Busy Output

Table 1-The dual-function *Hold* and *Ready/Busy* pin is programmed as a Ready/Busy output. Totem-pole and open-collector options are available.

tioned single-transistor EEPROM array, which is 4 or 8 Mb in the case of the NX25F040A and NX25F080A (\$7.79 and \$12.65 in 10k quantities). The company has announced plans for 1-, 2- and 16-Mb versions as well.

Because the chips are organized into 512 ('040) or 1024 ('080) 536-byte sectors, "K" means 1072 in Nexcom's case. Bargain bit shoppers should keep the -5% (i.e., 1072 vs. 1024) advantage in mind when comparing against other less "K"apable chips. The extra 24 bytes per sector are useful for tagging, time-stamping, error detection and correction, and the like.

The array is double buffered with twin 0.5-K (er, 536 byte) SRAMs. As you'll see later, they cache transactions, hide the EEPROM access time, and eliminate software machinations.

Fairly elaborate write protection helps keep your system from shooting itself in the foot. It starts with the *WP (Write Protect) pin, which disables all EEPROM writes. Even if writes are pin enabled, the device automatically powers up with writes disabled.

Assuming writes get past the first security checkpoints, they encounter on-chip logic that breaks the EEPROM into 16 blocks (i.e., 512 and 1024 sectors for the '040 and '080, respectively) which can be write protected in a top-down or bottom-up fashion.

The serial interface uses Motorola's SPI standard comprising a chip-select line (● CS), serial clock (SCK), and separate data-in (SI) and -out (SO) lines. Though data is normally shifted on the falling edge of SCK, there is a configu-

ration bit shown in Figure 2 to use the rising edge instead. The three speed grades offered don't refer to memory-access time but rather a SCK frequency of 8, 16, or 20 MHz.

Two more configuration bits define the function of the *Hold/*RB pin as you see in Table 1. As a Ready/Busy output (either totem pole or open collector), the pin signals whether the EEPROM array is busy or not and is handy to connect to a CPU interrupt or status input.

As a Hold input, the pin allows the CPU to temporarily suspend a command, rather than canceling it and having to start over. This feature is of most use when a higher priority task needs to perform a transaction to another IC over a multichip SPI bus.

Power-wise, the chips have a lot of bases covered, coming in 5-, 3.3-, and 3-V variants (all ±10%), not to mention a 2.7-3.6-V selection well-suited for battery operation. An on-chip charge pump generates the EEPROM programming voltage. Notice the unique AF bit in the configuration register that selects between nonharmonic charge-pump oscillator frequencies to minimize interference.

Active power seems reasonably low at 15 and 5 mA for 5- and 3-V versions, respectively. For reading, a special low-frequency command variant cuts power to a third if you limit SCK to 1 MHz or less. Of course, when the chip is deselected (and inputs aren't floating), power use falls into the few microamps range.

ACCESS-ONES

Table 2 details the command set that puts the chip through its paces broken into three categories-configuration and status, SRAM, and flash.

After powerup, the first thing your software should do is a reality check using **Read Device Information**. This command returns a read-only sector that includes part number, density, voltage, temp range, and other options. It's also a good idea to confirm or reset the previously described configuration register [which is writable, and thus suspect) using the **Re a d** and

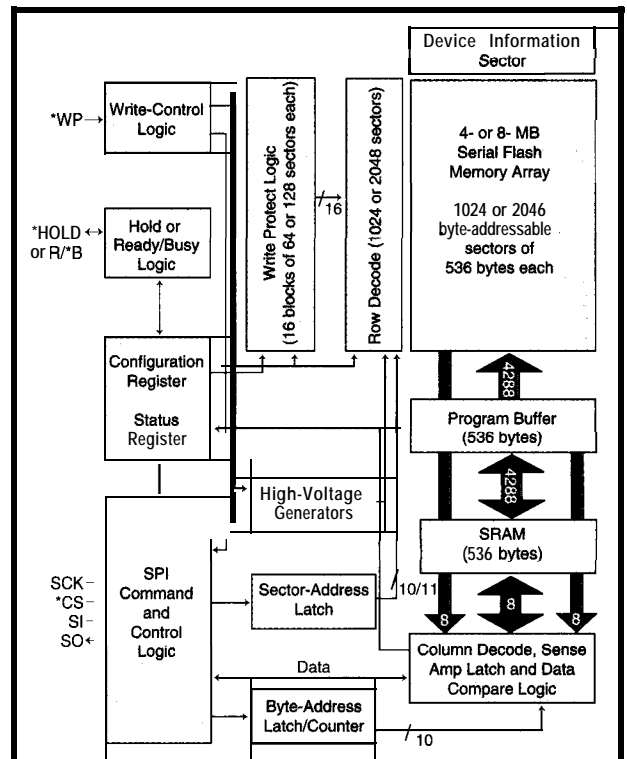


Figure 1-The Nexcom serial flash chips hide a big memory behind a small interface.

Write Configuration Register commands.

The chips also include a read-only status register (shown in Figure 3) accessed with the **Read Status** command. The Busy bit mimics the same-named pin function (i.e., it reflects the EEPROM array status). The TR (Transfer) bit performs a similar Ready/Busy function for the SRAM-related commands.

The WE (Write Enable) bit is manipulated with **Write Enable** and **Write Disable** commands to provide yet another tier of write protection. The CNE (Compare Not Equal) bit reflects

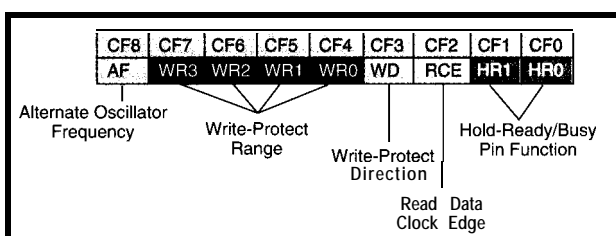
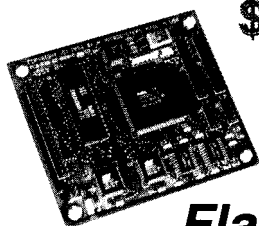


Figure 2-The configuration register is programmed with operating options using the **Read and Write Configuration Register** commands.

EMBEDDED DOS CONTROLLERS AT 8051 PRICES

Use Your PC Development Tools

NO MORE CRASH & BURN EPROM
Technology



\$195
Qty 1 Price

Flashlite

DOS Single Board Computer

with 572 k FLASH Memory disk drive

- ✓ 10 Mhz/8 Mhz CPU
- ✓ 2 Timers
- ✓ 512 k bytes RAM
- ✓ 4 Interrupt Lines
- ✓ 512 k/256 k FLASH
- ✓ 8 Analog Inputs
- ✓ 2 Serial Ports
- ✓ X-Modem File Transfer
- ✓ 24 Parallel I/O Lines

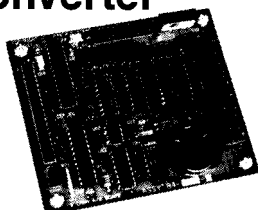
INCLUDES DOS & Utilities

USE YOUR TURBO C COMPILER OR
QUICKBASIC COMPILER
SAVE TIME, MONEY AND HEADACHES

A/D Converter

\$95

Qty 1 Price



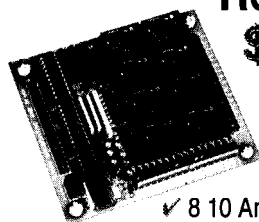
- ✓ 8 Channels, 12 Bits
- ✓ 6 μ s. Conversion Time
- ✓ Clock/Calendar Option
- ✓ Includes Drivers & Apps.

GET YOUR EMBEDDED CONTROLLER
PROJECT RUNNING FAST!
WITH THESE ACCESSORIES

Relay I/O

\$139

Qty 1 Price



- ✓ 8 10 Amp Relays
- ✓ 8 Opto-Isolated Inputs

JK microsystems

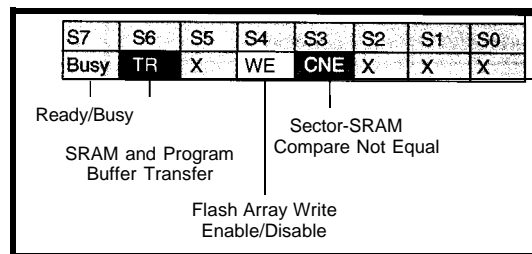
Cost Effective Controllers for Industry

TO ORDER (510) 2364151

IFAX (510) 236-2999—email: jkmicro@dsp.com

Visit our WEB site—www.dsp.com/jkmicro
1275 Yuba Ave., San Pablo, CA 94806

Figure 3—The status register discloses whether the EEPROM (Busy) or SRAM (TR) are currently preoccupied, the status of software write protection, and the result of the Compare Sector To SRAM command.



theresultofthe Compare Sector With S RAM command and is cleared with Clear Compare Status.

The SRAM commands access the dual buffers (SRAM and program buffer) and transfer data between them. Even though the program buffer can't be written directly, it's possible to shuffle data around (via *T r a n s f e r* commands) to coerce both 0.5-KB buffers into mimicking 1 KB of RAM when otherwise not being used for EEPROM transfers. Note that the Read and Write commands can start at any byte address and be of any length between 1 byte and the entire buffer length.

The flash commands are where the rubber meets the road. Like the RAM commands, both byte (i.e., Read and Write) and buffer (Transfer) variants are supported.

The double-buffering scheme comes into play for write commands (Write To Sector and Transfer SRAM to Sector). The contents of the SRAM are moved to the Program Buffer from

which the 5 ms ($V_{cc} = 5 V$) or 10 ms ($V_{cc} = 3 V$) EEPROM write is staged. In the meantime, the SRAM can set up the next write or revert to general-purpose use.

All the commands offer lots of flexibility, and choosing the best strategy depends on application characteristics, notably SCK frequency and whether transfers are byte or sector oriented.

For example, start with an EEPROM array write speed of 5 ms ($V_{cc} = 5 V$). Getting a entire sector into the chip takes a bit less than 5000 clocks [i.e., $536 \times 8 + \text{overhead}$].

Thus, if SCK is only 1 MHz or so, there's little gain from fancy buffering schemes since the bus is the limit. But, if you're doing lots of partial sector- or byte-level manipulation or your SPI clock is much faster, clever optimization of the memory can help a lot.

INFO KEY

I must admit, I was well into writing the article before I noticed some-

Command Name	byte 0	byte 1-2	byte 3-4	n-bytes			
Configuration and Status Commands							
Read Configuration Register	8B	0000	0000	0000	<i>read/busy</i>	configuration	
Write Configuration Register	8A	configuration	0000				
Read Status Register	83	0000	0000	0000	<i>read/busy</i>	status	
Clear Compare Status	89	0000					
Read Device Information	15	0000	byte addr	0000	<i>read/busy</i>	read data	

Serial SRAM and Program-Suffer Commands						
Write to SRAM	82	0000	byte addr	write data		00
Read from SRAM	81	0000	byte addr	0000	<i>read/busy</i>	read data
Transfer SRAM to Prog. Buffer	92	0000	0000	0000		
Transfer Prog Suffer to SRAM	55	0000	0000	0000		
Read from Program Buffer	91	0000	byte addr	0000	<i>read/busy</i>	read data

Serial Flash Sector Commands						
Read from Sector	52	sector addr	byte addr	0000	<i>read/busy</i>	read data
Read from Sector Low Frequency	51	sector addr	byte addr	0000	<i>read/busy</i>	read data
Write Enable	06	00				
Write Disable	04	00				
Write to Sector	F3	sector addr	byte addr	write data		00
Transfer SRAM to Sector	F3	sector addr	0000			
Transfer Sector to SRAM	F3	sector addr	byte addr	clock 00 per byte		00
Compare Sector with SRAM	86	sector addr	byte addr	0000	<i>read/busy</i>	XOR of data

Table 2—There are 18 commands that access the configuration and status registers, RAM (SRAM and program buffer), and EEPROM. All values are in hex, and italics indicate device output.

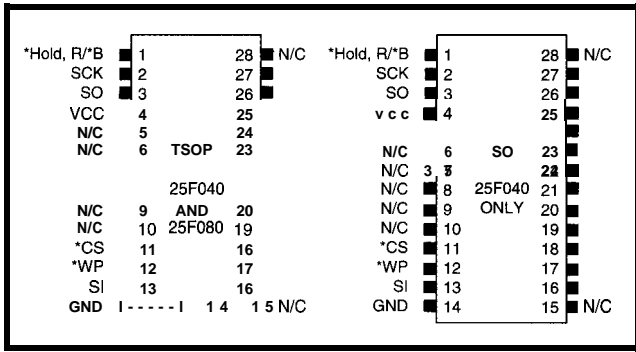


Figure 4—Mystery of the month: Why does an “8-pin” interface need a 24- or 28-pin package?

thing rather odd-namely, the packaging. See it for yourself in Figure 4.

Can't say I've ever seen a chip with more No Connects than signals, but there's always a first time. Having elaborated the premise that a pin-serial interface makes sense, somebody's got some explaining to do.

Sputtering x8 strategy retro-marketed with SPI! Top-secret double-die (one on each side) upgrade plan? Speedy test port? Somebody's brother-in-law having a 24-/28-pin fire sale?

Turns out to be nothing that exciting—just the simple fact that the die is too big to fit in the tiniest 8-pin packages. Not really a problem since all the N/Cs don't crimp your PCB layout.

I suppose you could even mount the chip on edge and leave 'em hanging in the breeze. Nexcom says they may offer downsized 16-pin versions of upcoming 1- and 2-Mb chips.

In fact, the IC packaging issue may be somewhat moot. Turns out, Nexcom is also offering the intriguing serial flash module shown in Photo 1. It takes advantage of a new 8-pin connector pioneered by ITT Cannon for GSM cellular-phone apps.

I must say the Nexcom module (\$13.50 in 10k quantities for the 8-Mb

unit) makes even recently downsized flash cards like CompactFlash and MiniCard (see “Flash Fight Flares,” *INK 76*) seem bulky.

If there's any bad news, it's that the module is small enough to end up with all the pens, lighters, and pocket knives in the black hole for the chronically misplaced detritus. Thoughtfully, the gadget includes a slot for a safety leash.

Notice the interesting pad layout in Figure 5—in particular, the dual-function WP/DT (Write Protect, Card Detect). On the host side, the WP/DT connection should be pulled up.

Then, card insertion can be detected by a low-going edge as the DT (and ground) connection is made. The card detect phase ends as the module slides past the DT land.

Subsequently, the WP/DT connection is interpreted as hardware write protect. If it's grounded on the module, write protection is enforced. Otherwise, it remains pulled up, signaling that host writes are allowed.

I did have a chance to fiddle with a module/connector combo and was especially pleased with its user-friendly nature. The skill required of the fumblefingers is minimal, and there's a satisfying detent-like tactile and au-

dible (clicking sound) feedback that minimizes ambiguity about whether the module is fully inserted or not.

Although insertion and removal force are low, the module seems to be gripped tightly enough to overcome the typical daily turbulence hand-held gadgets encounter. Like a Porsche shifter (I've heard), the arrangement just has “a good feel.”

NICE NICHE

While many applications are well served by the traditional choices (i.e., low-density serial EEPROM or high-density parallel flash), I'm sure there are situations where the Nexcom chips and modules make sense—everything from cellular answering machines and wireless fax to downsized data loggers.

Serial buses continue to gain popularity, and the high-speed capability of the Nexcom SPI port goes a long way towards defusing performance complaints. The cost of connecting to the chip is low, maybe close to zero if your system already has other serial-bus peripherals.

Contrast the SPI interface with the dozens of connections required for a parallel chip, not to mention the many more required by PCMCIA and its latest descendants. For those who need a bunch of bits in a small package, the Nexcom chips and modules may be just the ticket. □

Tom Cantrell has been working on chip, board, and systems design and marketing in Silicon Valley for more than ten years. You may reach him by E-mail at tom.cantrell@circuitcellar.com, by telephone at (510) 657-0264, or by fax at (510) 657-5441.

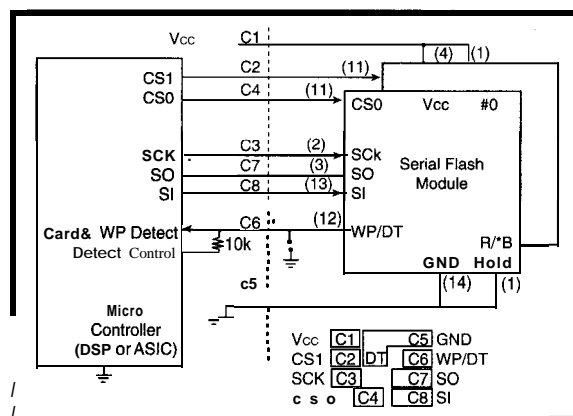


Figure 5—The serial flash module connector cleverly multiplexes card-detect and write-protect signals on a single pin.

SOURCE

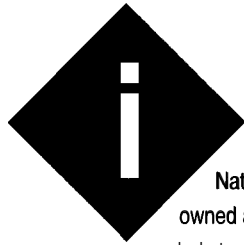
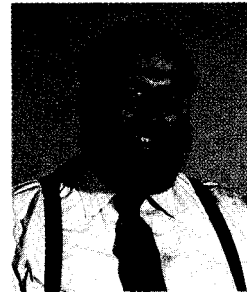
NX25Fxxx, NexFlash
Nexcom Technology, Inc.
532 Mercury Dr.
Sunnyvale, CA 94086
(408) 730-3690
Fax: (408) 720-9258

IRS

431 Very Useful
432 Moderately Useful
433 Not Useful

PRIORITY INTERRUPT

The Fast Track



had an interesting experience last weekend. I was invited to Watkins Glen, NY, for the Lysol 200 Busch Grand National NASCAR auto race. Even though I consider myself a sports-car buff, ratified by the fact that I have ever owned a few exotic cars, I've never felt a driving ambition to have my hearing shattered at a racetrack. Perhaps if I regularly tuned in to TNN instead of CNN, I might be more acclimated.

I went for two reasons. First, the invitation came from the primary sponsor of one of the racecars. When a guy throws a million dollars on the table so he can get up close and personal with burned rubber and gasoline, I figure he must have a screw loose or there really is something to all this. The second reason was curiosity. Given the super-integrated fly-by-wire metal-skinned rolling computer that I drive to the office daily, I could only speculate at the technical wizardry built into a **racecar** costing ten times as much.

As a sponsor's guest, I was given a Hot Pass identification card, which afforded me the same access level as the pit crew and driver! It meant that I could go virtually anywhere before or during the race. I could watch events unfold on TV from an air-conditioned conference room aboard the race-team truck, or I could be so close to that action that I'd have to be careful not to get my shoe size shortened by a passing race car.

I actually expected my curiosity to be anticlimactic. As an engineer, I looked at **racecar** efficiency as simply another closed-loop process control problem. I wasn't prepared for a situation described with more oxymorons than a government agency.

A racing team, especially like the one I was with, has both engineers and mechanics. When I was introduced as an "electronics guy," I was immediately invited to view the racecar's **onboard** telemetry system that constantly transmits important data such as pressures, temperatures, speed, and biomedical information back to the truck. With it, you could watch the driver's pulse rate increase as he approached a particularly dangerous curve. You could also see how really lousy gas mileage is at 140 mph.

"It's neat to see all this transmitted back here for analysis. I suppose the **onboard** computer takes care of all the real-time corrections and fuel injection?" I said, almost knowingly.

"Nope and nope. There's no **onboard** computer," he stated matter-of-factly. Realizing my shock, he smiled as he continued, "We look at the data, and then we get a wrench."

In actuality, it's a little more complicated than that. Most of these **racecars** have telemetry data systems and pit-to-driver radio intercoms. As I understand it, however, the telemetry data is for "informational purposes only." As a strict means of providing an even playing field, the rules allow only NASCAR-approved items on the car. These do not include fuel injection or closed-loop computer control. While the telemetry data may be used during practice sessions to determine the effects of tweaking specific systems, only the radio intercom can be used during the race itself.

The demanding rules seem to eliminate engine performance as the primary variable. Winning ultimately results from the team's ability to manually tune dynamic stuff like suspension and tire pressure, combined with an experienced driver who can keep it on the road.

One entertaining side note. Whichever network is broadcasting a race, it seems that the latest toy is the in-car TV camera. With it, TV watchers get a driver's eye view from specifically selected cars. It's not an altogether altruistic choice, mind you. Having the network mount one of these cameras in the car requires a substantial donation to their accounts-receivable department. When I announced plans to attend the race, one of the people at the office suggested that I specifically check out one of these in-car cameras. It seems that Ken, Jeff, and I designed the controller board used in the camera system.

In the end, it was a weekend of earsplitting noise. I went to Watkins Glen to satisfy my curiosity and check out the computers. It's ironic to discover that perhaps the only computer on the car during the race is something I had a hand in designing.

steve.ciarcia@circuitcellar.com