EMBEDDED PC MONTHLY SECTION

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

#89 DECEMBER 1997

DEBUGGING TECHNIQUES

The Zen of BDM

Remap Your Keyboard

Logic Analyzers—
Two Construction Projects

EPC: Meet EPC Design
Contest Winners

12
0 74470 75349 0

$3.95 U.S.
$4.95 Canada

guest TASK MANAGER

No Slowing Down

Since I only write one guest editorial a year, it always makes me nostalgic. I think about columnists who have come and gone. I recall frantic moments when a photo isn't in and we're shipping to press. I start to remember a long time ago when I had far fewer phone calls and interruptions, and I'm pining for the "simpler" life.

However, the reverie is short. In truth, I like action and am thrilled to have been a part of the changes that have taken place in *INK* over the three and a half years I've been on staff. It's good to see that companies now fully recognize that we have good readers who know a lot about designing things. Suddenly, they want to sponsor contests, advertise, send editorial.... Our pages fill up fast.

In 1997, probably the most gutsy thing *INK* did was sponsor the *Embedded PC* Design Contest. Although we had companies interested in advertising in *EPC,* the section was new. But, in the bold tradition of *INK,* we gambled. We decided that if anyone was going to lead the embedded industry it should be us. We knew that if we could organize a contest that made sense, you'd come through for us.

And you did. Take a look at the projects that won prizes. There are some very impressive, marketable products in that lot. I know two of the winners are already seeking corporate sponsorship of their designs. More power to them. It's to encourage this kind of enterprise that *INK* continues to support embedded design contests.

In 1998, Circuit Cellar *INK* will continue to look for opportunities for growth. In *Embedded PC,* for instance, while Rick Lehrbaum will keep on bringing us updates on trends with PC/104 and the embedded-PC world, he'll be joining us as an *EPC* feature author. Many thanks to Rick for a splendid job anchoring PC/l 04 Quarter.

What are we putting in its place? RPC-Real-Time PC-a column dedicated to helping you get to know all there is to know about real-time operating systems. We're placing this column in the *EPC* section since a high percentage of embedded-PC implementations fall into real-time control.

However, as always, we approach technology by offering a multiprocessor approach. If you make RTOSs for nonPC processors, we want to hear your angle on real-time issues as well. Just send your proposals and manuscripts in.

But enough on what's to come. Let's take a look at what's already here. Craig Haller kicks off this Debugging Techniques issue by giving an overview of on-chip debugging, while Ingo Cyliax zeros in on ColdFire's serial BDM interface. Frustrated with the limitations of low-cost logic analyzers, Janusz Mlodzianowski builds his own, and Cheng-Yang Tan, unwilling to have his keyboard settings dictated to him, remaps it for his own purposes.

Tom Napier opens a new MicroSeries on NCOs. He spends his first column discussing the manipulation of wave signals. Jeff makes magnetic field strength audible, and Tom checks into another Hot Chips conference.

In *EPC,* after I give you a glimpse of the *Embedded PC* Design Contest winners, Francis Deck introduces you to a high-speed logic analyzer for Windows 95. Rick closes PC/104 Quarter by showing how embedded PCs get bolstered to take on the environment, and Fred simulates a paper-tape reader for an industrial milling machine.

Janice

janice.hughes@circuitcellar.com

CIRCUIT CELLAR®

THE COMPUTER APPLICATIONS JOURNAL

EDITORIAL DIRECTOR/PUBLISHER
Steve Ciarcia

EDITOR-IN-CHIEF
Ken Davidson

MANAGING EDITOR
Janice Hughes

TECHNICAL EDITOR
Elizabeth Laurençot

ENGINEERING STAFF
Jeff Bachiochi

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
Rick Lehrbaum
Fred Eady

NEW PRODUCTS EDITOR
Harv Weiner

ART DIRECTOR
KC Zienka

PRODUCTION STAFF
John Gorsky
James Soussounis

ASSOCIATE PUBLISHER
Sue (Hodge) Skolnick

CIRCULATION MANAGER
Rose Mansella

BUSINESS MANAGER
Jeannette Walters

ADVERTISING COORDINATOR
Valerie Luster

CIRCUIT CELLAR INK®, THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (860) 6752751. 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 – Meadows Marketing

PRINTED IN THE UNITED STATES

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

Circuit Cellar BBS-24 Hrs.,2400/9600/14.4k bps, 6 bits, no panty. 1 stop bit, (860)871-1988. For information, mail to info@circuitcellar.com. World Wide Web: www.circuitcellar.com

INSIDE ISSUE 89

EMBEDDED PC

www.circuitcellar.com

READER I/O

APOLOGYTOREADERS

We at Dartmouth Printing Company extend our apologies to you for the unfortunate omission of editorial copy and the advertisement for R4 Systems that occurred within Jeff Bachiochi's article on page 76 of the November 1997 issue. The magazine you received is not representative of the quality that normally leaves our plant.

Tim Gates
Dartmouth Printing Company

Jeff's November column, "Nonintrusive Interfacing—Using Kid Gloves," and R4 System's advertisement are available in their entirety via the Circuit Cellar Web site in both viewable and downloadable formats.

Editor

BIG GUNS

I enjoyed Do-While Jones' "HDTV-The New Digital Direction" *[INK 86]*, but I had one small comment. The author said he doubted that it would be practical to build a CRT with 1080 guns. In fact, there are a couple companies doing just that with even more "guns," although perhaps in a different sense than he described.

A new technology-Field Emission-uses microscopic pyramids of emitters, multiple pyramids per pixel, a fraction of an inch from the phosphors. It uses x-y drivers to put a high voltage on a batch of emitters for a particular pixel which emit the electrons that are accelerated to the screen with another voltage (like to a regular CRT). This way, you have the potential for high resolution and high brightness in a flat screen. I'm not certain if any of this is in production yet, but I've seen several articles in *Electronic Engineering Times* about the technology.

Bob Bass
rpbassQionet.net

I was referring to a traditional CRT (a large vacuum tube with filament-heated cathodes, many inches from the screen). I agree that new technology, such as you describe, might find its way into living rooms some day.

Do- While Jones
do_while@ridgecrest.ca.us

BUT I ALREADY KNEW THAT!

During my Thursday class on "Fuzzy Logic Technology in Appliances" at this year's ESC-West, Constantin von Altrock put a viewcell on the overhead projector and said, "This is a great magazine in which to learn more about fuzzy-logic designs." It was Circuit *Cellar INK!*

So, at least 200 engineers saw your magazine cover that day! I had to smile to myself because the latest issue was waiting back in my hotel room for me to read that night.

Stephen **Buol**
Cedar Rapids, IA

AMPLITUDE ERROR

I enjoyed reading Mike Podanoffsky's article ("Compressing Audio and Video Over the Internet," *INK 86)* until I came to the section on audio PCM encoding. The a- and n-law codecs have variable *amplitude* encodings, not frequency. Higher frequencies tend to have smaller amplitudes, but the codecs don't inherently know that.

Analog or digital band-pass filtering is used to shape the spectrum being encoded. Offhand, I can't provide any specific references, unless you want to go back to the early Bell Labs Technical Journals of the late '50s and early '60s and the T1 carrier channel banks.

Roger J. **Pryor**
rjpryor@unix.infoserve.net

I stand corrected. The a- and p-law codecs have variable amplitude *encoding. The reference was my error in describing the compression. Thank you for pointing it out.*

Mike Podanoffsky
mikep@world.std.com

HDTV THANKS

Thanks for Do-While Jones' "HDTV—The New Digital Direction" *(INK 86).* It's the most informative article I've read in *INK* in a while. It did a good job of filling in some holes in my knowledge base. Keep up the good work!

Mark Nelson
markn @ tiny.com

STATEMENT REQUIRED BY THE ACT OF AUGUST 12, 1970, TITLE 39, UNITED STATES CODE SHOWING THE OWNERSHIP, MANAGEMENT AND CIRCULATION OF CIRCUIT CELLAR INK, THE COMPUTER APPLICATIONS JOURNAL, published monthly at 4 Park Street, Vernon, CT 06066. Annual subscription price is $21.95. The names and addresses of the Publisher, Editorial Director, and Editor-in-Chief are: Publisher, Steven Ciarcia, 4 Park Street, Vernon, CT 06066; Editorial Director, Steven Ciarcia, 4 Park Street, Vernon, CT 06066; Editor-in-Chief, Kenneth Davidson, 4 Park Street, Vernon, CT 06066. The owner is: Circuit Cellar, Inc., Vernon, CT 06066. The names and addresses of stockholders holding one percent or more of the total amount of stock are: Steven Ciarcia, 4 Park Street, Vernon, CT 06066. The average number of copies of each issue during the preceding twelve months are: A) Total number of copies printed (net press run) 33,283; B) Paid Circulation (1) Sales through dealers and carriers, street vendors and counter sales : 4,751. (2) Mail subscriptions: 22,810; C) Total paid circulation: 25,561; D) Free distribution by mail (samples, complimentary and other free): 4,824; E) Free distribution outside the mail (carrier, or other means): 150; F) Total free distribution: 4,974; G) Total Distribution: 30,535; H) Copies not distributed: (1) Office use leftover, unaccounted, spoiled after printing: 313; (2) Returns from News Agents: 2,435; I) Total: 33,283. Percent paid and/or requested circulation: 91.7%. Actual number of copies of the single issue published nearest to filing date are: (November 1997, Issue #88) A) Total number of copies printed (net press run) 37,000; B) Paid Circulation (1) Sales through dealers and carriers, street vendors and counter sales : 4,707. (2) Mail subscriptions: 20,317. C) Total paid circulation: 25,024; D) Free distribution by mail (samples, complimentary and other free): 8,399; E) Free distribution outside the mail (carrier, or other means): 0; F) Total free distribution: 8,399; G) Total Distribution: 33,423; H) Copies not distributed: (1) Office use leftover, unaccounted, spoiled after printing: 352; (2) Returns from News Agents: 3225; I) Total: 37,000. Percent paid and/or requested circulation: 90.3%. I certify that the statements made by me above are correct and complete. Sue Hodge, Associate Publisher.

NEW PRODUCT NEWS

Edited by Harv Weiner

DIGITAL THERMOMETER AND MEMORY

The DS1624 Digital Thermometer and Memory IC combines a digital temperature sensor and 256 bytes of EEPROM on chip to store temperature-related compensation information. The chip converts temperature directly, eliminating the need for an ADC, and it is calibrated at the factory. No external components are required, and the chip does not consume microcontroller resources. Applications include temperature-compensated crystal oscillators for test equipment and radio systems.

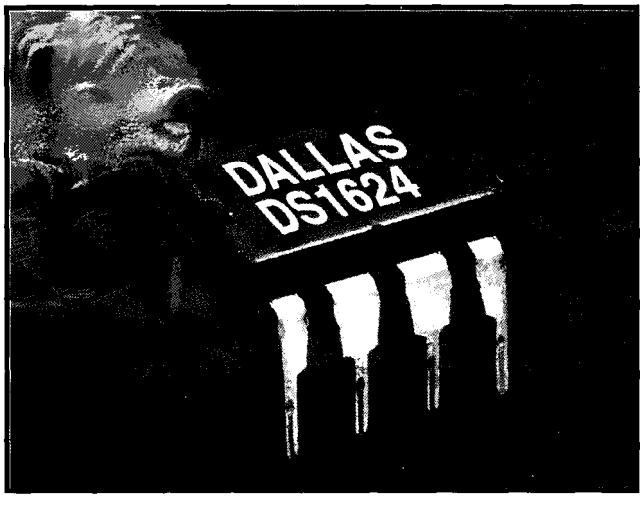
The thermometer provides 13-bit temperature readings (two-byte transfer), which indicate the temperature of the device. It measures temperature from -55°C to +125°C in 0.03125°C steps. Thermometer accuracy is 05°C across the 0-70°C range.

Temperature is converted to a digital word in less than 1 s and read or written via the popular two-wire bus architecture. This architecture features three-bit addressability, which permits users to multidrop up to eight chips along the bus.

The DS 1624 operates from 2.7 to 5.5 V and is available in either eight-pin PDIP or eight-pin SOIC packages. The PDIP package sells for $3.40 in quantity.

Dallas Semiconductor
4401 S. Beltwood Pkwy.
Dallas, TX 75244-3292
(972) 371-4448
Fax: (972) 371-3715
www.dalsemi.com

#501



DATA-ACQUISITION ADAPTER

The **LPT:*Analog! model 8c-I/O*** is a low-cost 12-bit data-acquisition adapter for the PC parallel printer port. The device features an 8-channel 12-bit ADC with user-selectable input ranges, two 12-bit DACs, and two 100-µA current sources for direct sensor excitation, as well as an uncommitted current mirror and multiple digital I/O lines.

The device is designed primarily as a universal sensor interface, and the supplied signal conditioning information makes it easy to collect data from analog output instruments, sensors, 4–20-mA loops, and many other analog signal sources at rates of up to 7500 samples per second. The DACs and digital I/O lines onboard are ideal for automatic sensor offset cancellation (auto-zero), as well as for real-time control applications.

LPT:*Analog!* is supplied with drivers, data-acquisition utilities, programming examples (QuickBASIC), and VB DLLs at no extra cost. The adapter sells for $99.

ADNAV Electronics
58 Chicory Ct.
Lake Jackson, TX 77566
(409) 292-0988
www.tgn.net/~adnav

#502

NEW PRODUCT NEWS

DATA-ACQUISITION MODULES

New **data-acquisition modules** from B&B Electronics can receive signals from up to eight external sensors, control various devices, and output analog voltages. The compact modules plug into DB-25 serial ports. Applications include monitoring sensors, controlling process and test equipment, and monitoring and controlling on/off states.

RS-232 and RS-485 modules have the ability to interface seven A/D channels, two digital input channels, one digital output channel, and four channels of eight-bit D/A outputs. Only four commands are needed to control the modules. For applications where long wire runs are required or a lot of line noise may be encountered, there are two 4–20-mA current-loop models.

Pricing for the four data-acquisition modules in the series ranges from $89.95 to $109.95. Each module comes complete with a demo program and API programs.

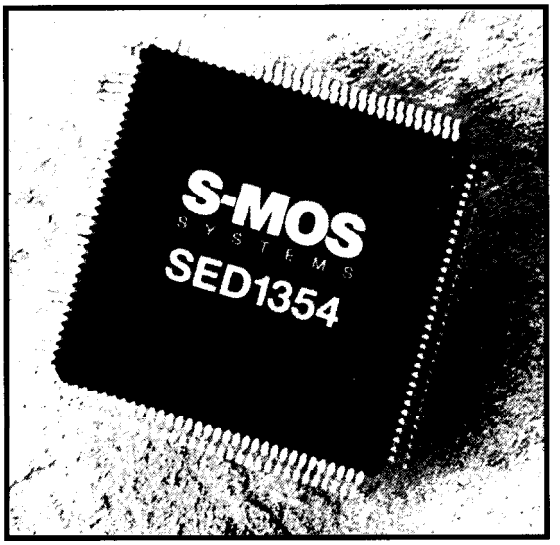
B&B Electronics Mfg. Co.
707 Dayton Rd.
Ottawa, IL 61350
(815) 433-5100
Fax: (815) 434-7094
sales8 bb-elec.com
www.bb-elec.com

#503

GRAPHICS CONTROLLER CHIP

The SED1354 is a low-cost, low-power color/monochrome LCD/CRT controller interfacing to a wide range of CPUs and LCDs. Its virtual display and split-screen capability is intended for embedded applications such as office automation equipment and mobile communication devices.

The SED1354 supports LCD interfaces with data widths up to 16 bits. Using frame rate modulation, it can display 16 shades of gray on monochrome LCD panels, up to 4096 colors on passive color LCD, and 64K colors on active-matrix TFT LCD panels. CRT support is handled via an external RAMDAC interface, enabling simultaneous display of both the CRT and LCD panels. A 16-bit memory interface supports up to 2 MB of EDO or FPM (fast page mode) DRAM.



Flexible operating voltages from 2.7 to 5.5 V provide for very low power consumption. Power consumption is reduced through the use of two power-down modes-one hardware and one software. Additionally, LCD power-sequencing signals are provided by the SED1354 to control an external LCD BIAS power supply, LCD backlight, and so forth.

The SED1354 sells for $9.70 in high volume.

S-MOS Systems
150 River Oaks Pkwy.
San Jose, CA 95134-1 951
(408) 922-0200
Fax: (408) 922-0238
literature@smos.com
www.smos.com

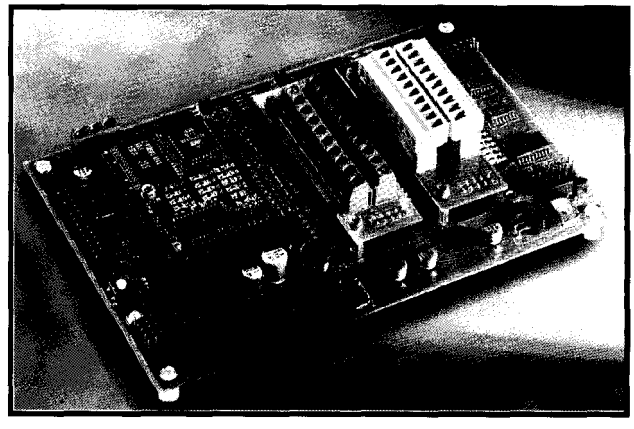
#504

NEW PRODUCT NEWS

C-PROGRAMMABLE CONTROLLER

The BL1700 is a C-programmable controller that's ideal for machine control, embedded systems, and OEM applications. It features 32 digital I/O lines (16 protected inputs and 16 high-current outputs), 10 A/D inputs, an 18-MHz processor, four duplex serial ports, LCD and expansion-bus ports, and DIN rail mounting. Up to 256 KB of nonvolatile flash memory facilitates in-system programming.

The BL 1700 includes conditioned analog inputs that interface directly with a variety of devices (e.g., photosensors, temperature and pressure sensors, and strain gauges). It also features digital outputs that directly drive solenoids and relays, and it can instantly add digital I/O, displays, relays, and D/A channels using the LCD and expansion-bus ports.



Programming the BL1700 controller is accomplished via Z-World's Dynamic C-a C programming language optimized for real-time, multitasking control. Dynamic C is an integrated software-development system providing an editor, compiler, and interactive debugger. The software comes with prewrittcn functions and software drivers.

The BL1700 is available with a development kit that includes a manual, schematic, programming cable, power supply, 128 KB of flash memory, and three standard wiring terminal blocks.

The BL1700 is priced at $199 in quantity.

Z-World
2900 Spaff ord St.
Davis, CA 95616
(916) 757-3737
Fax: (916) 753-5141
www.zworld.com

#505

NEW PRODUCT NEWS

COMPACT NETWORK SERVER

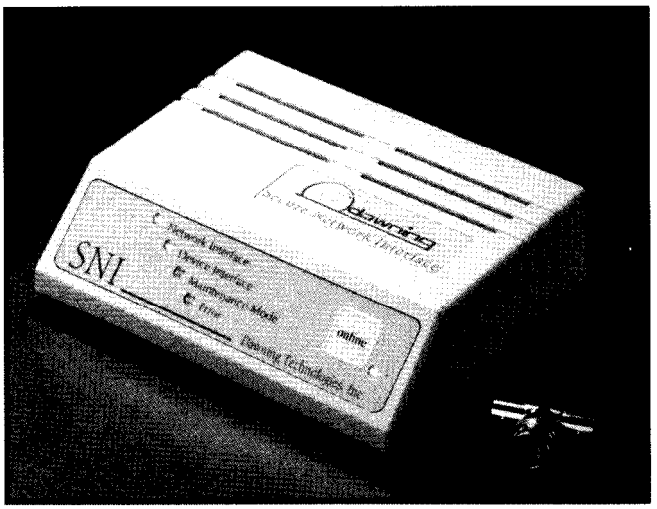
The Secure Network Interface (SNI) provides simple server-based real-time management of RS-232-compatible devices and Intranet/Internet networks. This Ethernet "micronode" has a discrete IP address, and the TCP/IP stack supports FTP, HTTP, SMTP, and telnet protocols. It provides many of the capabilities of larger servers and does not require a Mac or PC. The SNI is user programmable via enhanced BASIC using a remote HTML-compatible browser over the network.

Applications include creation of HTTP Web sites for serial devices, providing configuration and data-publishing capabilities over a network, replacement of bulky PC-based servers with an inexpensive and compact device, and remote access for troubleshooting. A demonstration is available at <weather.dawning.com>.

The SNI contains a 40-MHz16-bit 'x86-compatible processor and l-MB nonvolatile memory. It has an RJ-45 eight-pin RS-232 connection and is switch selectable for DCE or DTE communications. The network connection is 10 Mbps and can be either 10Base2 or 10BaseT. A 30-W wall transformer providing 12 VAC is included.

The SNI sells for $595.

Dawning Technologies, Inc.
409 Mason Rd.
Fairport, NY 14450
(716) 223-6006 • Fax: (716) 223-8615
sales@dawning.com • www.dawning.com **#506**



FEATURES

FEATURE ARTICLE

Craig Haller

The Zen of BDM

On-Chip Debugging

Still doing the old crash and burn debugging? Perhaps it's time to try your hand at debugging with on-chip resources. In Craig's opinion, it's the safest way to avoid intrusive overhead that alters the way your code runs.

Everybody's talking about BDM (Background Debug Mode). But, what is it really? What is the essence of BDM?

Strictly speaking, "BDM" is Motorola's term for a method of debugging and refers to a hardware port on their microcontroller chips. Other manufacturers use JTAG (IBM), OnCE (Motorola's On-Chip Emulation), MPSD (TI), and EJTAG (MIPS) ports, but many still call it BDM debugging.

For clarity, I refer to it as on-chip debugging (OCD). OCD includes various methods of using on-chip resources that enable complete software debug and aid in hardware debug.

In this article, I review OCD, what it is, and how to use it effectively.

WHAT'S OCD?

OCD is the latest addition to the debugger arsenal, which already includes the basic crash-and-burn debugging, hardware single stepping, ROM monitors and emulators, as well as in-circuit emulators (ICEs).

Early on-chip debuggers were debug monitors written into the microcode of the target processor (Motorola's CPU32). More advanced systems added features like real-time reading of the program counter [Analog Devices' SHARC processor) and near-real-time

reading of memory locations (Motorola's ColdFire).

OCD permits code download, reading and writing memory and processor resources, single stepping, processor reset, and status (running or halted). On-chip peripherals may be set to shut down during OCD (as opposed to while the chip is executing user code).

Some processors enhance OCD with other resources. IBM's '4*xx* PowerPC embedded processors have a seven-wire interface (RISCTrace) in addition to the OCD (RISCWatch), enabling a complete trace of processor execution. By capturing these lines in real time, a debugger can display a full trace of the last x instructions executed.

The new OCD standard for MIPS processors-Extended JTAG (EJTAG)—can retrieve a complete program trace as well as standard debug commands.

There are only a few drawbacks to OCD. The target usually needs RAM instead of ROM for debugging, and there is typically (not always) no form of real-time trace.

OCD HARDWARE AND SOFTWARE

In the general sense, OCD is a combination of hardware and software, both on and off chip.

On-chip OCD may be a microcode-based monitor (Motorola CPU32) or hardware-implemented resource (IBM PPC4*xx*). There may be resources available for the end user's code (e.g., breakpoint registers) or dedicated hardware (e.g., instruction stuff buffers), as with embedded PowerPC implementations.

OCD needs minimal external hardware. The chips and debugger host must communicate, often via a dual-row pin header and several pins on the processor.

The IBM '4*xx* and '6*xx* families use JTAG port pins in addition to reset, power sense, and ground, and connect via a 16-pin dual-row header, as shown in Figure 1. Motorola BDM uses five dedicated pins (sometimes multiplexed with real-time execution functions), power, ground, and at least one reset, all terminating in a IO-pin dual-row header.

Many DSP chips use a TI-style standard JTAG interface. Motorola expanded the interface's internal definition to include its DSP BDM equivalent, OnCE.

But, on-chip resources are only half the story. A target with an OCD processor and its dual-row header is useless without a host to communicate with.

The host runs the debugger software and interfaces to the OCD header. The debugger implements the user interface, displaying your code, processor resources, target memory, and so on.

The simplest hardware interface is a wiggler-a device that interfaces the parallel port of an IBM-type PC to an OCD header. It's both simple and slow. Other interfaces are serial port (RS-232) to OCD converters, high-speed parallel port to OCD, Ethernet to OCD, ISA-bus card to OCD, and others (see Figure 2).

Cost of host software runs from $49 to several thousand dollars. Hardware cost ranges from $100 to $5000.

BDM AS OCD

Motorola coined the term BDM with its CPU32 family of controllers, which was followed by the CPU16 family and ColdFire. These BDMs build on the concept of a ROM monitor and have a similar command set. The core of the hardware interface consists of a serial data in, serial data out, serial clock/breakpoint, and freeze status signal.

The commands are shifted into the chip serially and are 17 bits long. Table 1 lists the command set for the CPU32.

These commands closely mirror those used for years in ROM monitors. Single stepping is accomplished via hardware control of the BDM port or by software breakpoints in the codestream.

The processor is unaware of the BDM engine. It is not seen as an exception or interrupt. The background instruction BGND causes the processor to enter BDM. When GO is executed, BDM is exited and real-time code execution resumes.

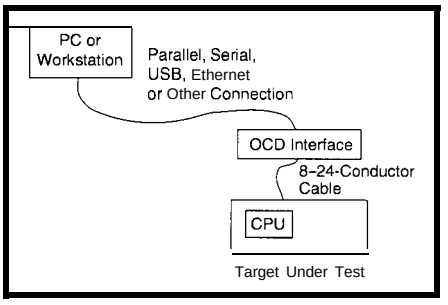


Figure 2—*OCD* interfaces come in many styles. *All* sit between a host-based debugger and the target.

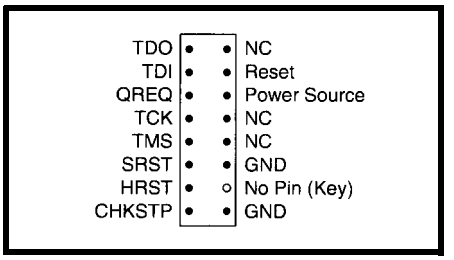


Figure 1 --This '6xx PowerPC *pinout* is *typical* of *OCD pinouts*. Most OCD processors have unique connections.

The embedded PowerPC BDM (Motorola MPC5*xx*, MPC8*xx*) works quite differently from the CPU32 type of BDM. The hardware interface is similar, but there isn't a specific command set.

Any serial stream entered is 7 or 32 bits in length (except for start, control, and length bits). The 32-bit bitstreams go into the instruction stuff register and come out the debug data register.

The host debugger stuffs PowerPC opcodes into the processor to be executed. This powerful design enables all system resources to be accessible, since the debug port has the same power as executing system code.

The 7-bit datastreams control on-chip breakpoint functions. Debug control registers exist to enable single stepping and other special controls.

The processor is aware of this BDM since it's a CPU exception. BDM may be entered on one of any number of exception-causing events (e.g., invalid opcode, address bus misalignment, nonmaskable interrupt, etc.) To resume real-time execution, the debugger stuffs a return-from-exception instruction (RFI) into the processor's instruction register.

ON-CHIP EMULATION

The OnCE interface on Motorola's DSP chip family enables the same types of debugging as the BDM interface. On most chips, the OnCE interface is implemented via dedicated pins. More recently, it's accessed via JTAG port pins as illustrated in Figure 3.

The OnCE port is more complex than the BDM port since it's a state machine controlled by the external debugger. Table 2 lists its capabilities.

JTAG DEBUGGING

The JTAG specification (IEEE 1149.1) is a method of doing full chip testing and was originally implemented to

enable testing of all the pin connections of a chip and its interconnections to other chips on the printed circuit board.

It's a serial protocol, and chips on the board may be daisy-chained together. In simple terms, the JTAG serial chain through the chip may be wired through any on-chip devices but minimally connects to all I/O pins and buffers.

The chain may be several score long to thousands of elements. There is no specification stating any inclusion of resources for software debug, nor is there a prohibition.

Various processors implement OCD via JTAG differently. The 600-series PowerPCs use the hardware test chain, which winds its way through many on-chip resources. Somewhere in the multithousand-stage serial chain is the instruction register, for example.

Debugging with this system is tedious since each core OCD action may take many trips through the entire JTAG chain. Although the debugger may only want a 32-bit piece of the chain, all elements must be traversed multiple times. Downloading user code may run less than 100 Bps (vs. over 40 kBps with other methods).

Another drawback of a shared hardware test/software debug chain is the way the chain is routed during chip design. Since this part of the design is typically the least critical, designers let the silicon autorouter lay out the chain's path after the rest of the chip is laid out.

Therefore, each revision may have a different JTAG chain, and the host debug software must be aware of every revision. TI often updates its OEM emulator software tool kit, but this approach doesn't help end users unless they have reliable debugger vendors.

An alternative method to the JTAG OCD is to use a different chain via the JTAG port. This approach is allowed for in the IEEE specification.

With this method, one chain is available for the hardware test and debug, another for software debug. This method is used in the IBM 400 series of PowerPC as well as in the Analog Devices SHARC DSP and the MIPS EJTAG-supported devices.

This secondary chain provides access to debug specific registers (usually only two or three are needed]. In IBM chips, the debug port has access to an instruction stuff buffer, debug control register, and debug status register.

The instruction-stuff buffer lets the debugger stuff any opcode into the core processor's instruction register, causing a single step to occur. By executing the proper instructions, any necessary action may be performed.

The debug control and status registers enable typical debug commands (e.g., single step and run). Since a separate chain from the hardware test chain is used, the chain length is less than 50 bits. There is some small overhead with each JTAG action to ensure that the proper chain is being accessed.

Command	Definition
RAREG/RDREG	Read address or data register
WAREG/WDREG	Write address or data register
RSREG	Read system control register
WSREG	Write system control register
READ	Read memory
WRITE	Write memory
DUMP	Read memory block
FILL	Write memory block
GO	Run CPU
CALL	Call user patch code
RST	CPU reset instruction
NOP	Null command

Table 1 -Some *OCD processors have dedicated command sets for software debug.*

Note that TI uses different flavors of the JTAG port on DSP chips. The C30 family has an MPSD port-similar to but not exactly like JTAG.

An advantage of the JTAG port for software debug is that it doesn't need additional pins on the processor for separate hardware and software debug. A disadvantage is the added overhead needed for each basic action.

DESIGNING A PROTOTYPE

There are many things to consider in designing your prototype target to take advantage of OCD capabilities.

First of all, use it! Some designers are so accustomed to ROM monitors or emulators, they ignore OCD features.

If possible, place the specified header on your board. If the prototype isn't the same PC card as the end product, then there's sure to be room on the board for the header.

If there isn't, look at the header specification from the manufacturer and debugger you plan on using. You'll probably find signals you don't need. IBM's RISCWatch header has several no-connect signals and a key [missing) pin.

Or, you can substitute a smaller header and make a conversion cable. The Motorola BDM for the CPU16/'32 family contains two ground signals and a DATA STROBE signal. Typically, one ground may suffice, and most debuggers don't use the data strobe.

The header doesn't have to be a dual-row header on 0.1" centers. Although this is the specification, feel free to modify it to fit your needs.

Next, be careful about the layout. It's helpful to keep higher frequency lines separated. If you're using a wiggler, there's no problem since the frequency doesn't usually surpass 100 kbps.

And, watch those traces. It's best to keep the OCD connector close to the CPU since the lines are typically not buffered. It's important to keep the traces approximately the same length, especially if they're serial communications lines.

For Motorola, these are DSI, DSO, and DSCK, and for JTAG interfaces, TMS, TDO, TDI, and TCK. I've seen problems, even with low-speed wigglers, when the lines meander around the board from the processor to the header, particularly with 3.3-V parts.

Remember, some JTAG ports are used for both hardware and software testing. The hardware use may necessitate connecting many chips on the board together via a JTAG daisy chain. This setup will greatly affect software testing and noise on the chain.

Also, watch the resistors. Motorola chips, in particular, set up the OCD configuration and access during hardware reset. It's important that the debugger correctly control these lines during this time.

Equally important is what happens when you test the board without a debugger. The manufacturer will also offer a recommended circuit for the OCD/JTAG header, which may or may not include resistor pull-ups and/or pull-downs. Other signals on the header may also have recommended circuits.

Virtually all OCD-equipped processors have multiple chip selects. One is typically configured during a hardware

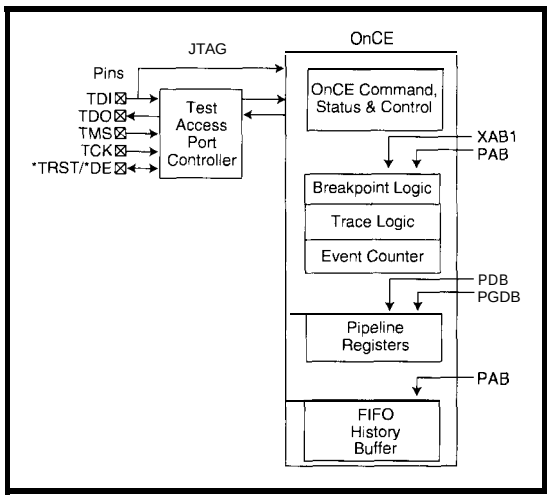


Figure 3—*Some OCD* engines *include* **sophisticated hardware to bridge** *the gap between OCD* **debugging and in-circuit** *emulators.*

reset to be used **with** whatever boot ROM or start-up code ROM is in the system. Many newer designs use flash memory for this purpose.

During debug, it's advantageous to use RAM instead of ROM to hold the code under test. You can have another chip select point to a bank of RAM.

The problem is that when you or the debugger cause a hard reset, the chip select for the RAM isn't appropriately configured. Also, you test code running with a chip select different from that in the final product-a situation that's better to avoid.

I recently developed a new solution. My boot chip select and general chip-select zero went to a 2 x 3 pin header. Onboard was a 128-KB flash memory and 128-KB SRAM. By changing the jumpers on the header, either device could be controlled by either chip select.

During initial debug, the RAM received the boot chip select. After code was burned into the flash (in the target, via an OCD flash programmer), the jumpers were changed, and the final debug and test were conducted.

You'll find if you socket your boot ROM, there may be a RAM with the same footprint that can fit in the socket during debug, or a simple socket adapter may be fabricated. Don't forget to make the socket writable.

Another common practice in a final product is to have the application code in as slow, small, and narrow a boot ROM as possible. This enables inexpensive storage.

The boot code sets up the hardware (minimally, the other chip selects) and copies the application code from the slower ROM to faster system RAM. These tasks are followed by a jump to the start of the application.

This kind of simple boot program is easy to implement. You may want to have the boot section of the code written and placed into a ROM on the boot chip-select line. During debug, the application code would not be in the ROM but still on the host.

When you reset the target under test, you execute the beginning of the boot code to set up the hardware and then return to the OCD. Now, your application under test may be downloaded to the RAM on the chip select it will run with in the final system.

There are tools on the market that let you program flash memory while it's on the target board. These tools work through the on-chip debugger.

By configuring the flash so the processor can write to it, programming it becomes easier. You may need to run a WRITE line to the chip and/or add 12 V controlled by a port pin (preferred) or a jumper. This technique may involve adding a trace or two to the PC board—definitely worth the added copper.

DESIGNING YOUR PRODUCT

There are many reasons to have access to the OCD in a final product. With the proper host support, flash-memory programming, production-line testing, and in-field debug are all possible. Even if you don't use it after production starts, the lack of access to OCD, if it is needed, may be costly.

It's not nearly as important as with the prototype to use the factory-specified header. But, watching the traces is as important, if not more so. Your product may be in a less friendly environment (e.g., electrical noise) that you may not be able to control as easily.

Watching the resistors is important, too. You want to ensure all start-up parameters are correct, which is especially important with Motorola's BDM interfaces.

Finally, setting up flash memory for writability is crucial. The ability to easily program flash-on the production line and in the field-will prove invaluable. And, you can eliminate the use of sockets for EEPROMs.

CHOOSING A DEBUGGER

First, consider the invasiveness of debuggers (i.e., the amount of system setup the debugger does for the user). A ROM monitor typically does some setup. An OCD debugger doesn't have to do this but often does. Why does this matter?

If the debugger does any setup and your code doesn't reproduce it in the exact way (and possibly at the exact time), your code won't run in the same environment as when it is tested. This is a perfect example of why your code will work with the debugger but not directly out of ROM.

Whether an OCD debugger has to do setup depends on your hardware configuration. Other similar invasions are the initialization of general registers, setup of an oscillator PLL, and so forth.

There are different thoughts on how much the debugger should protect the user. A common target has a bank of RAM into which your code is loaded for testing.

Assume your code is running in real time and goes into the weeds. If there's an errant pointer, you're now executing out of uninitialized RAM, which is garbage code. The code may go for a while, wreaking all sorts of havoc, until the debugger somehow regains control. This problem is tough to debug without a large trace buffer.

Alternatively, the debugger could have filled memory with some specific instruction before downloading your code. If this instruction is a **BREAK, BGND, TRAP,** or **INTERRUPT** that the debugger recognizes, a break would occur at the first errant instruction.

Should debuggers automatically do this? What about interrupt vector tables? Should debuggers fill in uninitialized vectors and trap on their use?

Some of these issues are easier to deal with, depending on the target chip. The embedded PowerPC chips have many options for protecting the user.

By setting bits in a register, you can cause the OCD mode to be entered for various events (e.g., execution of an unrecognized opcode, misaligned data fetch, etc.). If the debugger secretly sets these bits, you're debugging in a different environment than the one your code runs in. This is probably OK, but does your debugger give you access to these bits?

OCD SPECIFICS

A handful of OCDs are on the market, ranging in price from freeware to several thousand dollars. They all provide the basics-read and write registers, read and write memory, download code, single step, run, and so on. Most have source-level debug capabilities. Some work only with assembly code, others with any language.

Of most concern are the situations I mentioned. Is there hidden initialization? Are there user-friendly traps? And what about that start-up stuff?

Assume you ignore my suggestions for prototype design. Your target has its boot chip select attached to some type of ROM chip. Since this is debug time, there is no code in ROM yet.

The debugger is connected to the OCD header, and you want to start testing code. You have the debugger reset the target and then download your code. Wrong!

On reset, the only properly set-up chip-select line will be the boot chip select, and it's pointing to useless ROM. Whichever chip select is attached to your RAM must be initialized. But, by who (or what]?

Some debuggers have built in setups for known hardware. Usually, you can describe your custom target via dialogs to tell the debugger how to set up the board.

Others let you write command files (e.g., macros, scripts, etc.) to do the setup. These files have commands such as **WRITEL 0x1234, 0x5678,** which writes a **LONG** value of hex 1234 to location hex 5678.

With some debuggers, you must explicitly run the command file every time you reset the processor. Others do it automatically.

Again, the problem is that your code is now in an environment that's different from the reset environment, and your code didn't cause this change. If the only command is a setup of the RAM chip select, this problem probably isn't too big. Probably.

Another set-up issue is target-processor speed. Many new processors use an inexpensive 32-kHz crystal with an on-chip PLL to boost the system frequency.

On reset, the PLL is at some default value, possibly a slow one. Often, your application's initialization code sets the PLL to a faster value, but during debug, this only happens after your code downloads.

If the debugger doesn't do any setup (hidden or not) and you do a download (via the boot chip select), the processor is most likely running at a slow speed. This slows your download.

All OCD protocols are implemented serially. The maximum OCD speed is usually a function of the CPU clock speed (about one-third or one-half the CPU speed).

Most OCD hardware interfaces start at a slow speed since the processor speed usually can't be determined. If the interface speed isn't set for maximum (either the fastest the CPU can handle or the interface can run, whichever is slower), the debugging speed is affected.

This situation is most obvious in the download speed of code. Some debuggers let you modify the interface speed in a command file. You'd do this only after you set the PLL speed, of course.

You probably have to set the interface speed to be slow at the start of the command file. Why?

Once you reset the target processor, it runs at its default speed. If it's slow, you must slow the interface to do your PLL setup and then speed up the interface.

Ideally, you may have a macro that runs whenever you hit the debugger RESET TARGET button or a command on your debugger that resets the target CPU, lowers the OCD speed, sets the processor PLL for desired speed, and raises the OCD speed to as fast as the processor allows.

Interrupt/break into debug mode on program-memory address
Interrupt/break into debug node on data-memory address
Interrupt/break into debug mode on an on-chip peripheral access
Enter debug mode using a DSP instruction
Read/write any DSP core register
Read/write peripheral memory-mapped registers
Read/write program or data memory
Step one or more instructions
Trace one or more instructions
Save or restore current chip pipeline
Read real-time instruction trace buffer
Exit debug mode

Table 2—*Motorola's OnCE engine offers* a powerful command set *for debugging.*

ONWARD!

Use this information to ask questions of the vendor, and see the debugger in use. Does it work with your favorite compiler? How does it communicate with the target? What is its invasiveness, and are those items fully documented!

I'm prejudiced about debuggers. I've written and marketed several-from basic DOS assembly-language-based debuggers to complete Windows-based high-level systems. But, I'll leave you to your own devices.

Good luck, and good debugging. ❑

Craig Haller is president of Macraigor Systems, an OEM of embedded systems debug tools, and a firm believer that silicon manufacturers aren't marketing the advantages of OCD nearly enough. You may reach him at craig@macraigor.corn.

SOURCE

Debuggers
Macraigor Systems, Inc.
P.O. Box 1008
Brookline Village, MA 02147
(617) 739-8693
Fax: (617) 739-8694
www.macraigor.com

I R S

401 Very Useful
402 Moderately Useful
403 Not Useful

Serial BDM Interface for ColdFire

FEATURE ARTICLE

Ingo Cyliax

Since conventional debugging techniques often don't work on embedded systems, Ingo checks out ColdFire's on-chip debugger. He builds a simple BDM-to-serial interface that helps him control and modify the chip.

In my three-part MicroSeries *(INK* 86–88), I describe a MC68030-based workstation used by computer-science undergrads at Indiana University to learn more about computer architecture.

While this platform served us well for several years, I'm always looking for new architectures and technologies to enhance this lab. For the students, gaining experience with current technology is almost as important as learning the fundamentals when it comes to finding a job after graduation.

In my search for a new architecture to replace the MC68030 workstation, I'm currently evaluating the Motorola ColdFire architecture. In particular, I've been looking at the MCF5204 and MCF5206 implementations. Both are integrated microprocessors and include I/O and SRAM on chip.

Besides the ColdFire CPU core and 512 bytes of SRAM, the MCF5204 has one serial port, two timers, eight bits of general-purpose I/O, and a flexible 8-/16-bit bus interface. The MCF5206 adds a 32-bit bus interface, a DRAM controller, one more serial port, and an I^2C two-wire device bus interface. These chips can implement a complete microprocessor system with only the addition of a boot PROM.

The ColdFire is well-suited for use in our lab because it's a 32-bit processor based on the 68000 architecture. Motorola has reduced the complexity by only implementing the most frequently used instructions and addressing modes. So, the ColdFire core is very lean, which is important if you target the low-end embedded 32-bit market where the cost/performance ratio is crucial.

Since the ColdFire core and chips are targeting the embedded-systems market, conventional debugging techniques are inadequate most of the time. Embedded systems usually don't have the luxury of a keyboard and display system, and many don't even have a serial port.

This situation makes software-based debugging not feasible or often intrusive. In a "hard" real-time system, software-based debugging techniques can't be used, since any additional software overhead may alter system response time.

Imagine a motor controller that synthesizes the AC waveforms used to drive a motor. A well-designed system would have enough processing power to perform this task and stay cost effective, but it might not have sufficient power to run a software debugger, which responds to breakpoints and traces sections of the code.

In embedded applications where software debugging is not feasible due to performance issues or I/O restrictions, you'd typically use an in-circuit emulator [ICE). An ICE emulates the signals and timing of the CPU and replaces the CPU in the system under test. It gives an external debugging host a window into the system by allowing real-time traces and access to the state of the CPU.

The ICE uses a pod that plugs into the processor's socket, thus replacing the processor, and a cable that connects the pods to the ICE interface. The interface unit is usually fairly bulky.

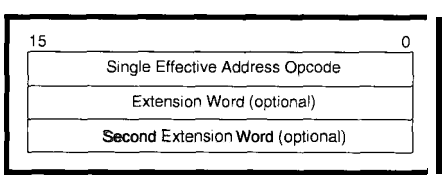


Figure I--The *ColdFire instruction format consists of one 16-bit base opcode with one or two extension words, depending on the address mode used. Most register-to-register movements and ALU operations can be specified using the single-word instruction format.*

However, in this article, I examine the on-chip debugging features of ColdFire. I also want to show how you can build a simple BDM (Background Debug Mode) to serial interface that enables the user to control the processor and examine and modify its state.

With this interface, it's also possible to read and write memory in the target system while the CPU is running. But first, let's take a brief look at the ColdFire architecture.

COLDFIRE ARCHITECTURE

ColdFire is a variable-size instruction RISC processor. Many of its RISC-like instructions that operate on registers are encoded as short 16-bit instructions, while instructions that use immediate data or addressing information use extensions to the short instructions. Figure 1 depicts the general instruction format.

The 68000 has always implemented variable-sized instructions, and this fact can be demonstrated by showing that ColdFire binaries run on a 68030 processor, but not the other way around. The 68030 implements many more instructions and instruction forms, and also has an MMU and FPU coprocessor.

Like the 68000, ColdFire has eight general-purpose data registers and eight address registers. It differs from the MC68000 series in that it only has a single stack pointer.

The ALU in the ColdFire is a leaner version of the 68000 ALU, so it doesn't implement all the 68000's operations and data types. All these reductions have made ColdFire so lean that it can now be synthesized from a hardware-description model. The ColdFire core is portable across different chip technologies, so its architecture enables optimal chip-level integration.

Table 1--Here we can see all the commands which are implemented in the ColdFire BDM. Each command can take input data, output data, or both.

Command	Data In/Out	Description
218x	—/2	Read CPU data register
208x	2/—	Write CPU data register
1900	2/1	Byte read
1940	2/1	Word read
1980	2/2	Long read
1800	3/—	Byte write
1840	3/—	Word write
1880	4/—	Long write
1D00	—/1	Byte dump
1 D40	—/1	Word dump
1D80	—/2	Long dump
1 coo	1/—	Byte fill
1C40	1/—	Word fill
1 cao	2/—	Long fill
0C00	—/—	Go
0000	—/—	NOP
2980	2/2	Read CPU control register
2880	4/—	Write CPU control register
2D8*x*	—/2	Read BDM register
2C8*x*	2/—	Write BDM register

BACKGROUND DEBUGGING MODE

The BDM module is a hardware-based debugging module that's embedded on chip. It has connections to the CPU core and the internal CPU bus. Since the BDM module sits so close to the CPU, it has essentially the same view of the system as the CPU.

This setup is important if the CPU core is embedded in a chip with on-chip resources and cache. It's quite difficult to debug systems with cache turned on.

But, BDM isn't exactly new. It was first implemented on Motorola's CPU32 core and appears in most MC683*xx* embedded processors (e.g., '68332 and '68360). It can also be found in the CPU16 core, which shows up in the MC68HC16 series of processors.

By using a simple serial interface, the BDM module permits an external debugger to examine the state of the CPU, read and write memory, and control the execution of the CPU. Extra status signals provide detailed information about the CPU's state in its various execution phases.

The original BDM on the CPU32 core was implemented in the CPU's microcode. And, the CPU would have to be in a halted state before BDM could become active.

This situation occurs when the CPU executes a HALT (BGN D) instruction or encounters a catastrophic condition (e.g., a double bus fault] or when the external ● BGND pin is asserted by the debugging interface. In all BDM implementations, asserting the *BGND pin while resetting the CPU causes the CPU to enter a halted state before fetching the reset vector and initial stack pointer.

On ColdFire, BDM is implemented in a separate hardware module and runs in parallel to the CPU. Therefore, by stealing bus cycles, the BDM module can read and write memory while the CPU is running.

An external debugging system can also monitor the CPU state through the BDM interface. Real-time debugging features are implemented in the BDM module via programmable 'lard-ware-trigger facilities.

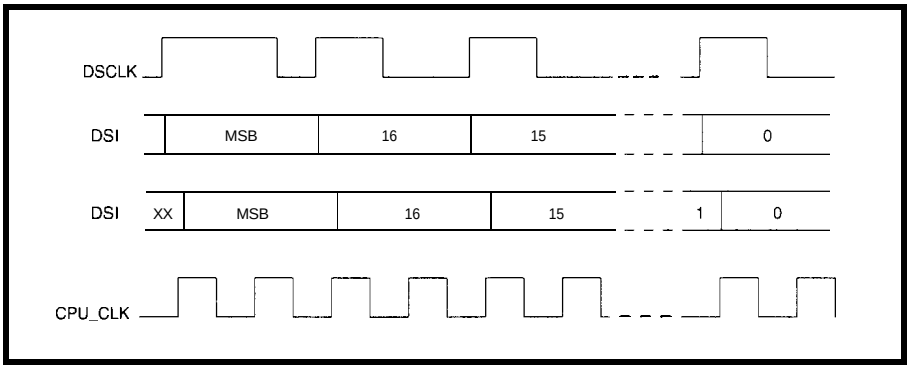


Figure 2—*The serial protocol* uses *three wires (DSI, DSO, and DSCLK)* and *the CPU clock to transfer 17-bit* words between *the* debugger and *BDM* module in the CPU. *The DSI* and *DSCLK signals have to be* synchronous to the *CPU_CLK* and meet set-up and *ho/d times specified in the ColdFire reference manual.*

PROTOCOL

The basic BDM protocol consists of a three-wire interface (i.e., DSI, DSO, and DSCLK). The external debugger transfers data by simultaneously clock-

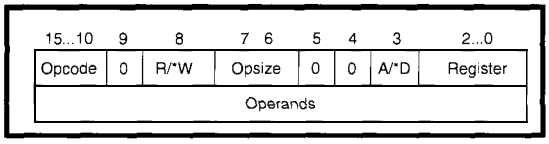


Figure 3-A *BDM* instruction specifies the data sizes requested and *the command code to perform.*

ing data in and out of the chip using the DSI and DSO lines. The DSCLK can run at any speed between DC and half the CPU clock.

Commands and responses are 17 bits long and shifted most significant bit first. With the most significant bit-the S/C bit-the CPU indicates if the response is an error or not. The debugger always clears this bit on all transfers. Figure 2 shows the basic timing and word structure of the BDM protocol.

Commands and data are encoded in these17-bit words. The BDM module decodes these commands and performs the required action. The commands sent to the BDM have the general format shown in Figure 3 and can be simple commands, requiring no data, or complex commands, requiring data to be sent to the BDM. Commands that read registers and memory also respond with data from the BDM module.

There are three types of transactions-command only, command plus data, and command plus data plus response. BDM overlaps the response from a phase with the command/data for the next phase. A sophisticated debugger might be able to optimize these overlaps to squeeze some extra performance from the interface.

In particular, it could send the next command while reading the command-completion code from the last command. A simple debugger may just send a N 0 P command whenever it reads the response from the BDM.

To execute any command that can alter the CPU registers or state, the CPU must first be halted by asserting the *BKPT pin on the BDM interface. The processor status signals (PST[3:0]) indicate $F when the CPU halts. The debugger can also read the BDM status register at any time and determine the state of the CPU and BDM module.

Once the CPU is halted, the debugger can examine and alter any CPU and BDM register. Memory can be read or written at any time. Table 1 shows all the BDM commands available in ColdFire.

REAL-TIME DEBUGGING SUPPORT

There are two facilities in the ColdFire BDM that aid debugging in real-time environments. The hardware-trigger facilities enable generation of breakpoints and external trigger stimuli, and the tracing facility gives a window into the state of the CPU while it is executing.

A hardware trigger can be generated on memory references, program counter locations, and operand data values. These can be combined to generate cascaded triggers (i.e., one trigger arms the second trigger) or simple triggers.

Once a trigger occurs, the response can either cause the CPU to halt or generate an exception to the CPU or a noop. The noop function is useful if you only want to trigger an external oscilloscope or logic analyzer.

The BDM module also routes internal signals to the outside using the PST and DDATA interfaces. A configuration register can be programmed to specify the kind of information displayed on these signals.

The PST interface monitors processor status, enabling an external monitor to trace the internal CPU activity. Table 2 shows the encoding for the internal CPU state on these signals.

The DDATA interface provides additional information. Depending on the CPU state, it may present data such as operand values or branch locations. Also, the processor can send data to this interface using special debug instructions. DDATA with the PST interface can give an accurate view of what's going on inside the CPU.

PST[3:0]	Definition
0000	Continue execution
0001	Begin execution of instruction
0010	
0011	Enter user mode
0100	Begin PULSE or WDDATA
0101	Begin execution of taken branch
0111	Begin execution of RTE
1000	Begin l-byte transfer on DDATA
1001	Begin 2-byte transfer on DDATA
1010	Begin 3-byte transfer on DDATA
1011	Begin 4-byte transfer on DDATA
1100	Exception processing
1101	Emulator-mode entry exception processing
1110	Processor is stopped waiting for interrupt
1111	Processor is halted

Table **2**—*The PST[3:0]* signals provide a window info the inner workings *of the ColdFire CPU. Each code signals a state in the core.*

SIMPLE SERIAL BDM INTERFACE

Let's see how to build a simple RS-232-based BDM interface using a Basic Stamp II and a PLD. The interface lets any serial-based workstation and even a terminal control a ColdFire CPU through its BDM interface.

I chose a Basic Stamp since it's readily available and easy to prototype with. The interface isn't fast, but it's functional.

Since the ColdFire BDM module is implemented in hardware, it requires all of its input signals to be synchronous to the CPU system clock. This task is easily accomplished by using the flip-flops in the PLD to synchronize signals to the CPU clock. Simple D-type registers can be used, too.

The Stamp generates the 'BKPT signal, which halts the CPU, and the *RESET (hardware reset) signal. The Stamp also generates the DSCLK and DSI signals, while monitoring the DSO line.

I also wired in the PST[3:0] to some spare signals on the Stamp. Even though the signals change too fast for the Stamp to trace them, they can be used to detect if the CPU is halted.

This information is redundant, since the command and status register (CSR) in the BDM also contains it. However, it's more efficient for the Stamp to read a four-bit nibble from the I/O pins to determine the CPU state than to execute a serial BDM command to read a register.

The Stamp communicates with the debugging host/terminal over its RS-232 interface using the TX and RX pins. Care must be taken that the ATN

signal is not driven low by DTR on the Stamp, which puts the Stamp into its programming mode. I added a jumper to isolate the signal during normal operation.

The Stamp and external circuitry get their power over the BDM interface connector from the CPU. The interface schematics are shown in Figure 4.

The software is organized bottom up. First, the routine bdm (see Listing 1) takes the data in bdmsnd and transmits it using DSCLK and DSI. At the same time, it records the response from the CPU to the last transfer over DSO and records it in the variables bdmrcv and bdmmsb.

The next level up executes the serial BDM command primitives. read 1 in Listing 2 implements reading a 32-bit word from memory. It takes two 16-bit words-a d d r h and a d d r 1 -to specify the memory location to read, and it records its response in da t a h and datal.

To perform a long-word read, the Stamp sends the value $1980, followed by the two address words, using bdm. After sending the command, I wait for the BDM to perform the read operation by stealing deadtime on the bus.

The Stamp does this by sending a N 0 P command ($000) and looking at the response. It continues until the response is either valid data ($0xxxx) or a bus error ($10001). The first data word is stored in da t a h, and another NOP is sent to read the second word to be stored in datal.

Other BDM commands (e.g., go) and reading/writing registers are implemented the same way.

At the top level of code, a command interpreter reads a command packet from the host over the serial port. After decoding it, it calls the appropriate BDM subroutine to execute the BDM commands necessary to implement the request and collect its response.

The response gets encoded properly to send it back to the host over the serial port. To inform the debugger of the CPU state, it displays the current PST[3:0] encoded as a hex digit as part of the command prompt.

The command interpreter reads line-oriented text commands much like a debugging monitor would. Thus, a debugger can use the interface with a dumb terminal program, maybe even remotely over a modem connection. The Reference section directs you to the PBasic program for this project.

With this interface, I can now control the execution of the ColdFire core and read and write CPU registers and system memory. I can also control the BDM registers to program triggers and DDATA[3:0] sources.

Except for the execution speed, this little gadget is quite useful. A faster implementation of this BDM interface could be based on a PIC CPU programmed in assembly or C, which would also make it inexpensive-always a bonus when working in academia.

Listing 1—*bdm* is a low-level function that clocks out the 17-bit command word and clocks in any data sent from the BDM in the ColdFire chip. All the bdm subroutines implemented in the serial BDM inferface use this function to communicate with the ColdFire BDM.

```
' perform one BDM transaction
' send 17 bits starting with '0' and the word  "bdmsnd"
' receive 17 bits-S/C in bdmmsb and the rest   n bdmrcv
bdm:
  dsi = 0                          ' MSB is always  low on sending
  pulsout       pdsclk,1           ' clock it out
  bdmmsb = dso                     ' read MSB
  ' send 16 bits of cmd  and record 16 bits of  status at  same time
  for i = 0 to 15
  dsi  =  bdmsnd  >> (15  i)
  pulsout pdsclk,1
  bdmrcv = (bdmrcv << 1   | dso
  next
  dsi = 0                          ' restore every :hing
  dsclk = 0
  return
```

INTERFACING WITH GNU GDB

I introduced the GNU source-level debugger (gdb) in Part 3 of my Micro-Series (*INK* 88). It can use a serial port or the network to remotely debug processes running on a different system. It can also be extended to implement various protocols needed to talk with ICEs or debugging monitors.

gdb also has a generic remote protocol interface. I'm currently working on interfacing gdb with my serial BDM interface. A serial-based BDM debug-

Listing 2-r *e a d 1 is a typical function implemented in the serial BDM interface. It sends the "read long" command and address and receives the long data, which is stored at that location by sending NOPs.*

```
readl:                             read  long  memory
  bdmsnd = $1980                   ' send read long command
  gosub bdm
  bdmsnd = addrh                   ' high word of address
  gosub bdm
  bdmsnd = addrl                   ' low word of address
  gosub bdm read1  again:
  bdmsnd = $0000                   send NOP  to  read  status
  gosub bdm
  ' loop while waiting for BDM to complete transaction
  if bdmmsb = 0 then read1  done
  if bdmmsb = 1 AND bdmrcv = 0 then read1 again
  serout 16,n19200,["Error  in  readl",
  hex bdmmsb, hex bdmrcv,cr,10]
  return
  ' BDM is done and has sent high word of data
  read1  done:
  datah = bdmrcv
  bdmsnd = $0000                   ' send NOP  to  read  low  word
  gosub bdm
  data1 = bdmrcv                   ' record low word of data
  return
```

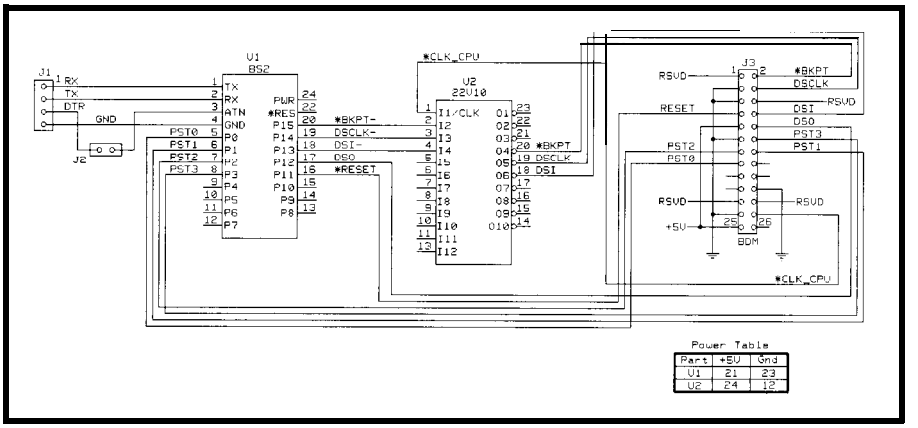


Figure 4—*This simple* RS-232 *BDM* interface uses a Parallax *Basic Stamp II and a PLD. Check the References for information on obtaining the PLD JEDEC file and Stamp code for this interface.*

gcr module that works with a terminal and source-level debugger is useful.

WHAT ELSE?

The BDM module is a powerful debugging feature of ColdFire-especially during system and program development. It can also reduce manufacturing costs, since this interface can be used to perform in-circuit tests of a system during assembly and burn-in. It also enables in-circuit programming of flash- and EE-based memory for program and configuration and parameter storage.

The BDM module makes ColdFire a good candidate for our architecture lab. We can build a minimal system consisting of a MCF5204 or MCF5206 and some useful I/O for experimentation.

BDM can then bootstrap the processor and write the students' code into SRAM or external DRAM. Such a system offers all the essentials for our class (e.g., interrupt-driven I/O, timers, etc.).

And since ColdFire chips are cost-sensitive and the BDM interface can be made cheaply, it's feasible for students to buy a ColdFire module that they can take home. GNU software could be bundled with this system on a CD-ROM, enabling students to do development on their own PC.

Ingo ***Cyliax works as 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.*

REFERENCES

Text

Motorola, *MCF5200PRM/AD Cold-Fire Programmer's Reference Manual,* Phoenix, AZ, 1995.

Internet

MC5206 Evaluation Module bundle, www.hh.avnet.com/marketing/coldfire.html.

GNU tools (gdb), prep.ai.mit.edu/pub/gnu.

Stamp code and PLD JEDEC file for the serial interface, ftp.cs.indiana.edu/pub/goo/SerBDM.

SOURCES

Basic Stamp II

Parallax, Inc.
3805 Atherton Rd., Ste. 102
Rocklin, CA 95 765
(916) 624-8333
Fax: (916) 624-8003
info@parallaxinc.com
www.parallaxinc.com

ColdFire

Motorola
MCU Information Line
P.O. Box 13026
Austin, TX 7871 l-3026
(512) 328-2268
Fax: (512) 891-4465
www.mot.com/SPS/HPESD/prod/coldfire

I R S

404 Very Useful
405 Moderately Useful
406 Not Useful

FEATURE ARTICLE

Janusz Młodzianowski

A Simple Multipurpose Logic Analyzer

There are times when what's available on the market is not what you want. The price, features, or expandability just doesn't meet your current needs. What do you do? If you're Janusz, you build your own.

We all know that debugging a digital circuit with a single logic probe or scope can be frustrating. What we really need is a multichannel logic analyzer.

Dedicated analyzers are available on the market, but the price (in excess of $1000) is too high for small-scale designers. The price of smaller PC add-on card analyzers isn't much lower.

Some manufacturers offer simple devices capable of capturing up to eight signals via a PC parallel printer port. Although such analyzers cost less, their functionality is limited. The maximum sampling frequency is on the order of a few tens of kilohertz.

Since I've designed digital circuitry for a number of years now, I concluded that the only sensible solution would be to build my own logic analyzer. I needed a simple yet expandable instrument with at least 16 TTL probes, capable of capturing data at about 50 MHz and connected to a PC via a bidirectional parallel printer port.

I didn't want to use any exotic components and microcontrollers. The hardware had to be built with off-the-shelf components and fit a short 3U Eurocard. Photo 1 shows the result.

THE BASICS

A logic analyzer is basically a multichannel probe that can read, store, and process a number of digital data vectors.

In the measurement phase, the data is clocked into the analyzer's internal storage RAM. While reading, each captured vector is compared with a preset trigger pattern.

When the comparator matches data with the trigger vector, measurement ends. The captured data transfers to the host PC and is presented to the operator.

Figure 1 illustrates the hardware, including the modules for the PC interface, logic probe with input register and data storage, programmable comparator, and control circuitry. Figure 2 shows the timing diagram and the key signals controlling the hardware.

Once all measurement parameters are set up, the measurement cycle is initiated by a momentary pulse on the START line, which sets the BUSY line active. Once BUSY is asserted, low-to-high transitions on the WR374 line registers in the input buffer the data vector present on all probes.

The WR374 strobe originates from the internal time base clock or the analyzed circuitry. In Figure 2, this clock (internal or external) is shown as INPUTCLK. Signals WR4040 and WRAM are derived from WR374.

When the data vector is registered, the falling slope on WR4040 advances the RAM address generator. The following rising slope on WRAM stores the data in the analyzer's internal RAM. The analyzer is then ready to accept the next vector.

Once the data vector is registered, it is compared with the trigger word. When a match is detected, the TRIG line asserts high and decrements the preprogrammed Event Counter. When the Event Counter reaches zero, the second preprogrammed counter-the Delay Counter-is enabled.

Once enabled, the Delay Counter counts WR374 pulses. When it reaches zero, its output signal (DELAYTC) ends the measurement cycle.

The end of the measurement is signalled by BUSY reverting to an inactive state. Strobes WR374, WR4040, and WRAM are disconnected from the source. The measurement cycle can

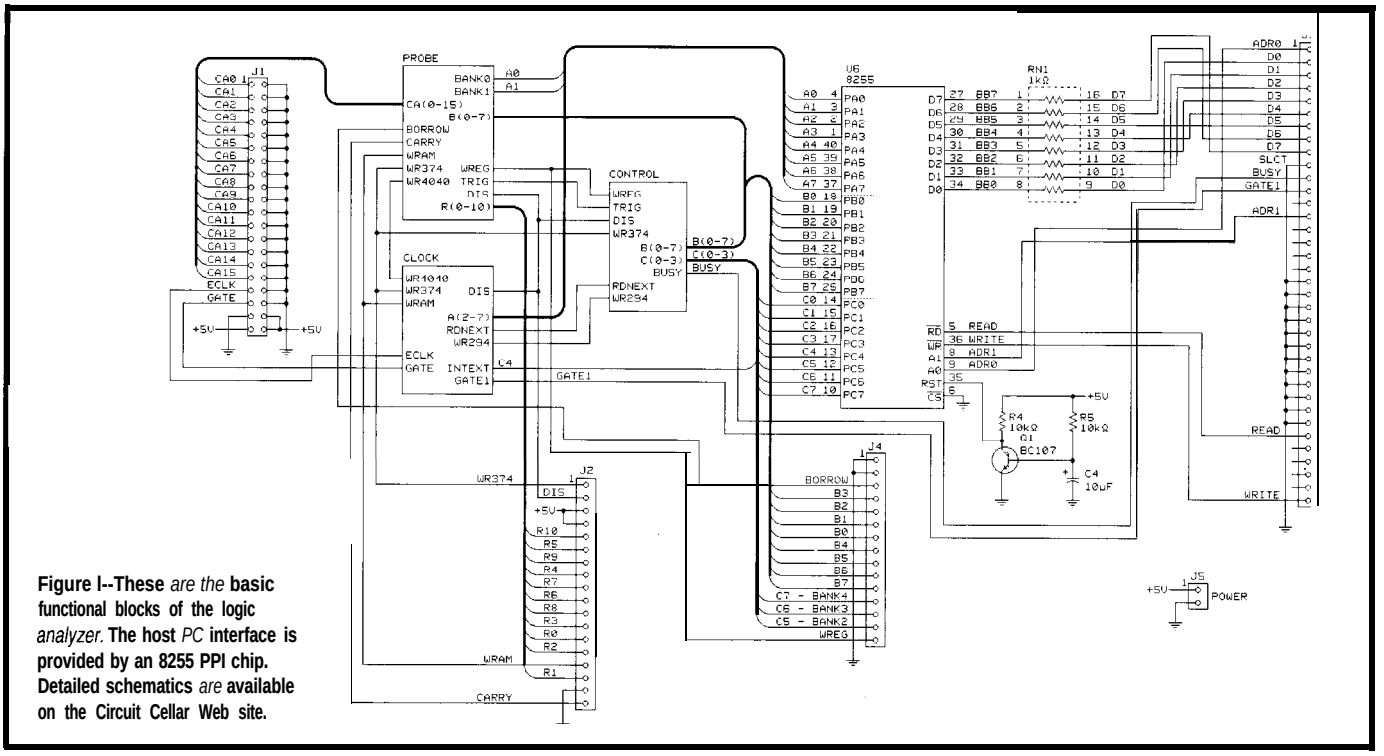


Figure I--These *are the* **basic functional blocks of the logic** *analyzer.* **The host** *PC* **interface is provided by an 8255 PPI chip. Detailed schematics** *are* **available on the Circuit Cellar Web site.**

also be stopped by asynchronously pulsing the STOP line.

Once the measurement cycle ends, the data stored in the RAM can be read back to the host PC. While reading the data, the RAM address generator is advanced by pulsing RDNEXT.

Let's examine the functions of the Event and Delay Counters. The analyzer's objective is to detect a particular pattern in the registered data. Usually, the operator is interested in the first instance of the vector (e.g., the first call to int10 after reset in a PC).

But sometimes, you might want to catch the tenth call to int 10. The Event Counter then skips the first nine calls and is preprogrammed to 10. An Event Counter feature is not present in the simple analyzers I know of.

The Delay Counter enables the hardware to capture a given number of data vectors after the final trigger. Suppose I want to see up to 50 processor machine cycles after the tenth call to int 10. The Delay Counter should then be set to 50.

A closer analysis of the timing diagram shows the measurement cycle ends after two more vectors (e.g., 52 in this example) are captured. This can be compensated for with the software.

In this design, the RAM operates as a ring buffer. When the entire RAM is filled with data, the counter wraps around and the process continues.

It doesn't matter where in RAM the data is stored. When the measurement cycle ends, the RAM address generator points to the first location after the last registered vector.

The ring-buffer approach enables the use of some simple counting arithmetic. When the Delay Counter is preset to 0, no additional samples are taken after the last trigger, so the buffer contains the maximum count of samples acquired before the last trigger. When the Delay Counter is set to maximum, the RAM contains the maximum count of samples after the last trigger.

From the user point of view, the delay count set in the Delay Counter can be positive (i.e., after the trigger) or negative (i.e., before the trigger). The count equal to zero means the buffer contains one half of the buffer size samples before and after the last trigger.

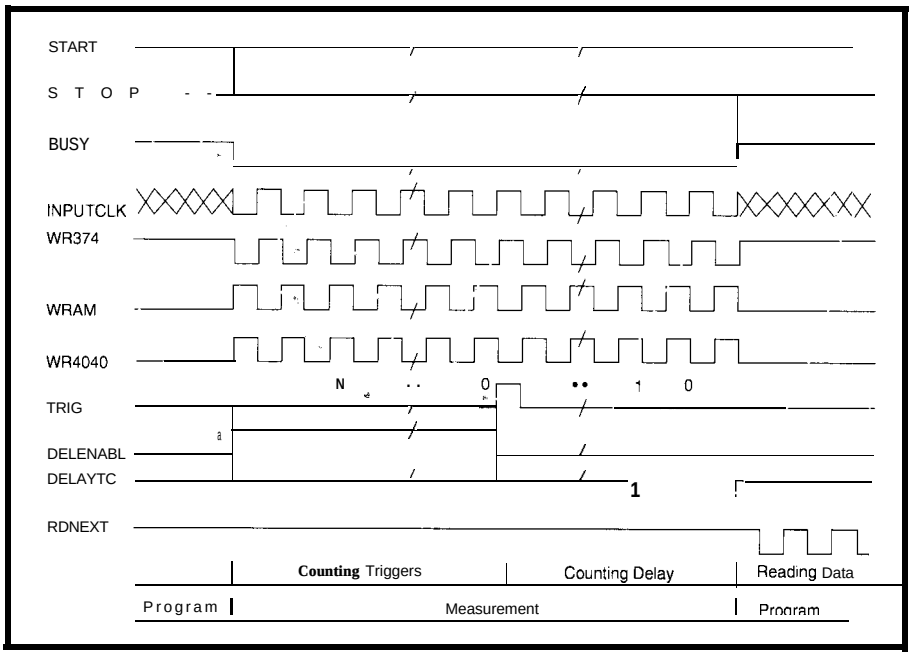


Figure 2—*From* **this** *timing* **diagram, you see that the hardware operates in two** *modes—Measurement* **and Program.**

Data Port (Base+O)							
D7	D6	D5	D4	D3	D2	D1	D0
D7(9)	D6(8)	D5(7)	D4(6)	D3(5)	D2(4)	D1(3)	D0(2)
Status Port (Base+I)							
D7	D6	D5	D4	D3	D2	D1	D0
BUSY(11)	ACK(10)	PE(12)	SELECTED(13)	ERROR(15)	X	X	X
Control Port (Base+2)							
D7	D6	D5	D4	D3	D2	D1	D0
X	X	INOUT	INTR	SLCTIN(17)	INIT(16)	FEED(14)	STROBE(1)

Table 1 —*The PC Centronics printer port is controlled by three registers. Here are the signal names and corresponding pin numbers.*

HOST-PC INTERFACE

The analyzer connects to a host PC via a bidirectional parallel printer port. As you know, the printer port consists of three consecutive I/O addresses starting at the base address 0x278, 0x378, 0x2BC, or 0x3BC.

These addresses arc known as LPT ports. Offset 0 denotes a data port, offset 1 a status port, and offsct 2 a control port. Table 1 lists the signal and pin assignments. Newer bidirectional printer ports use an additional direction bit (D5) in the control register.

D5 equal to low sets the data port in output mode, while D5 equal to high sets the data port in input mode. Bit D4 in control register enables (HI) or disables (LO) the generation of interrupt request on level 7 whenever ACK changes from active to inactive.

The PC parallel printer port can serve as a general-purpose bidirectional I/O port. To make the printer port versatile, a simple data transfer protocol mimicking ISA PC input and output cycles is provided. A simple i of u n c library (see Listing 1) provides the support functions.

Lines INIT and SLCTIN serve as 'READ and . WRITE signals. On power-up, the BIOS sets both signals high.

Lines FEED and STROBE serve as address lines ADR1 and ADRO. So, up to four I/O registers can be addressed.

The 1 ptw r i te () cycle starts with the Printer Data port being set up as output, and the address lines ADR1 and ADRO set. The output data is then placed on the Printer Data port, and the * WRITE line is pulsed low.

The 1 pt rea d() cycle starts with the Printer Data port being set up as input with the valid address on lines ADR1 and ADRO. When 'READ is brought low, the byte present on data lines is read back to the PC. After the transfer, *READ is reversed high.

The printer status lines can be used as a five-bit input port. The practical USC of the printer port is somewhat complicated by the fact that FEED, SLCTIN, STROBE, and BUSY lines are inverted on the I/O connector. The i ofu n c software corrects thc polarity of these inverted lines.

The host PC sees the logic analyzer as one S-bit bidirectional data port, 16 control bits, and three status bits. To simplify the design, the 8255 PPI chip, operating in mode 0, implements all the necessary I/O registers.

Port B serves as a bidirectional data port, while ports A and C provide all control signals. Table 2 presents the description of the logic analyzer's registers and bit functions.

The PPI's ● CS is tied to ground, so the chip is constantly enabled. To further protect the 8255, 1-kΩ resistors are placed in series on all data lines. The analyzer's BUSY line is connected to the printer's BUSY line, and the signal GATE connects to PE. SLCT pulls down ● ACK, enabling the software to determine whether the analyzer is connected to the PC.

To help with programming the PPI, the i of u n c library is extended by functions directly accessing all 8255 registers. Control, data, and status signals arc available on a 36-pin Amphenol connector with the pin arrangement compatible with a standard Centronics printer. The system is powered by +5-V stabilized, calculator-type power supply.

LOGIC PROBE

The logic probe consists of a fast 16-bit input data latch and the mechan-

Listing 1—*The purpose of i o fun c is to access the logic analyzer through the IBM PC bidirectional parallel printer port. Only three sections of i o fun c are shown here. Remaining sections are available via the Circuit Cellar Web site.*

```
//include <dos.h>
#define DATA          0x378      /* assume LPT2 */
#define STATUS        ox379
#define CONTROL       0x37A
void lptwrite(char address, char value);
char lptread(char address):
char lpt5input(void);
void lptwrite(char address, char value)
{
  char adr=(address & 3)^3;
  outportb(CONTROL,0x4+adr);
  outportb(DATA,value);
  outportb(CONTROL,0xC0+adr);
  outportb(CONTROL,0x4+adr);
}
char lptread(char address)
{
  char temp:
  char adr=(address & 3)^3;
  outportb(CONTROL,0x24+adr);
  outportb(CONTROL,0x20+adr);
  temp=inportb(DATA);
  outportb(CONTROL,0x24+adr);
  return(temp);
}
char lpt5input(void)
{
  char tmp;
   tmp=inportb(STATUS);
   return((tmp^0x80)>>3);
}
```

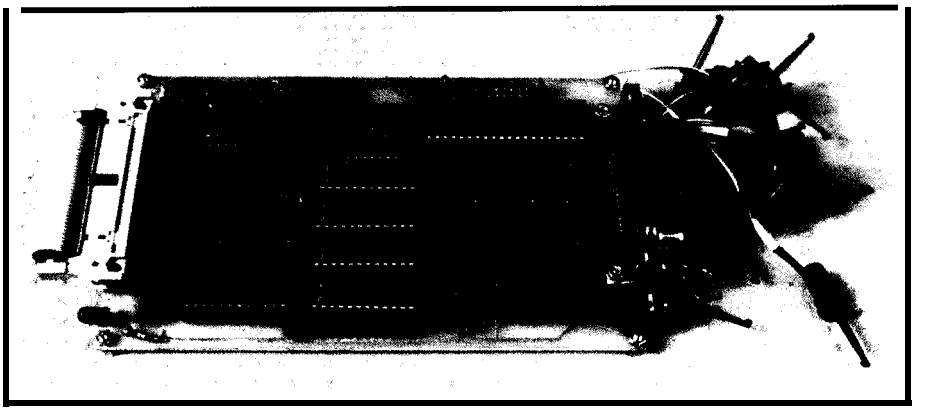


Photo 1 --The logic *analyzer* **is assembled on a double-sided, plated-through** *PCB.* **You can** *also see the* **ribbon** *cable with clip-on connectors.*

ical connection to the measured circuit. Connection is handled via a 30-cm ribbon cable and either individual clip connectors or one 16-DIL IC clip-on connector on one side and a 50-pin IDC connector on the other.

To simplify the design, the input probe doesn't have any analog input buffers. As the input signals are directly connected to two 74F374 registers (U7 and U16), the logic analyzer operates with TTL levels only.

The input data is clocked in by the WR374 signal. The DIS line (inverted BUSY) tristates the input buffers when the data is read back to the PC. Registered data is transferred into two fast 2-KB SRAMs (U12 and U20) strobed by the WRAM signal.

The RAM write cycle time limits the maximum operating frequency of the analyzer. With a RAM access time of 25 ns, the frequency is 40 MHz.

The RAM address is generated by the 74HCT4040 (U8) binary counter clocked by the WR4040 signal.

Using the 74HCT4040 limits the RAM size to 2048 bytes. The RAM address can also be changed by toggling the RDNEXT line.

COMPARATOR

The programmable comparator circuitry is the most important and complicated part of the design. The comparator can detect any of 0, 1, or X (don't care) states.

To distinguish between the three possible logic states, this "one-bit comparator" needs two reference bits (see Table 3). X*n* = 0 forces the comparator to signal the positive bit test regardless of the input value.

The trigger pattern is stored in four serial-in/parallel-out 74HCT164 (U10, U14, U18, U22) shift registers clocked by the WREG signal. The bit pattern is delivered via the data bit BO.

The comparator circuitry is burned into four 20V8 fast GALs (U9, U13, U17, U21). These GALs also serve as tristate RAM data output buffers/multiplexers.

Both the trigger register and comparator can be daisy chained for a longer data word. The BORROW input is used to lengthen the comparator. Low on the BORROW line enables the comparator. When the analyzer uses only 16 channels, a jumper must be placed between the BORROW and GND lines on pins 2 and 3 on the J4 connector.

Data Port							
D7	D6	D5	D4	D3	D2	D1	DO
D7	D6	D5	D4	D3	D2	D1	DO
Command Port A							
A7..A4	A3	A2	A1	A0			
FREQ	GATE	SLOPE	BANK1	BANK0			
Command Port C							
c7	C6	c5	c4	C3..C0			
BANK4	BANK3	BANK2	INTEXT	COMMAND			
Status Bits							
s7	S6	s5	s4	s3	s2	S1	so
BUSY	SEL	GATE	X	X	x	x	x

Table 2—*The logic analyzer is programmed* **using the &?-bit bidirectional** *data bus, 16-bit control, and 3-bit status ports. The registers are accessed via the parallel printer port using i o fun c h calls.*

Sn	Xn	
0	0	don't care
0	1	low
1	1	high

Table 3—*To implement the comparator, two reference bits per channel are needed.*

The CARRY output extends the length of the trigger register. The low-to-high transition on the TRIG output of the comparator signals the detection of the trigger pattern. TRIG also decrements Event Counter in the control circuitry.

Lines BANK1 and BANK0 open Probe's output buffers and select the particular RAM bank to be read. Signals BANK4..BANK2 can be used when more RAM banks are present.

Exercise great care when activating BANK*n* lines. When multiple lines are active or PPI port B is in output mode, GAL output buffers can be damaged.

CONTROL CIRCUITRY

The heart of the system is the control circuitry. It consists of the Command Register, Time Base Clock generator, 256-bit Event Counter, and 204%bit Delay Counter.

The Command register (74HCT138, U5) is controlled by COMMAND bits in Command Port C. In total, eight commands can be generated (see Table 4).

The heartbeat of the logic analyzer comes from the programmable Time Base Clock. Control bit INTEXT from Command Port C selects between internal or external clock source.

While the external clock is selected, the SLOPE bit from Command Port A selects the active slope of thc clock signal applied to the ECLK pin on the probe's IDC connector. When the internal clock is selected, the uscr chooses between one of the 16 settings derived form the internal 32.76%MHz crystal-controlled oscillator.

The SN74294 (U1) serves as a programmable frequency divider. The division ratio set up as FREQ bits in the Command Port A is written into the SN74294 by the WR294 signal.

The frequency division ratios available from this chip vary from 2^2 to 2^{15}. The D flip-flop (U2A) and the multiplexer buried into 16V8 (U4) GAL facilitate the division ratios of 1 and 2.

The external clock and all signals derived from it (i.e., WR374, WR4040, and WRAM) can be temporarily suspended by an external signal applied to the GATE pin, which is also available as the analyzer's status bit. The logic level that inhibits the clock can be selected by the GATE bit in Command Port A.

The Event Counter (74HCT40103, U23) is programmed by the ECWR Command. It counts TRIG signals generated by the Comparator. Its output is registered in the U1 la flip-flop, which enables the Delay Counter decremented by WR374.

The Delay Counter (U15, U19) is programmed by HIWR [high byte) and LOWR (low byte) lines. Its output (DELAYTC) strobes the 74HCT74 flip-flop, which resets the BUSY signal.

Both Event and Delay Counters have been implemented using HC or HCT technology. The popular CMOS parts have low maximum frequency. START and STOP lines asynchronously set and reset the BUSY flip-flop.

When the analyzer is disabled (BUSY is inactive), the logic burned into U4 disables WR374 and WRAM. WR4040 is reconnected from the clock to the RDNEXT Command bit.

SOFTWARE

My logic analyzer is of little use without the software. In fact, the hardware's usefulness lies in its software.

The hardware is controlled via the kernel, which consists of 17 functions. The kernel functions enable the user to set up all measurement parameters,

Command	Action
START	Initiates Measurement cycle
STOP	Stops Measurement cycle
WREG	Writes trigger word pattern
RDNEXT	Reads data word from internal RAM
WR294	Sets internal frequency generator
LOWR	Writes Low byte to Delay Counter
HIWR	Writes High byte to Delay Counter
ECWR	Writes byte to Event Counter

Table *4-Eight commands control the hardware of the logic analyzer. Each command toggles one particular strobe line.*

initiate and stop the measurement cycle, and read the internal RAM.

Listing 2 shows the kernel header file. The library is written in Borland's Turbo C.

The byte transfers between the PC and analyzer are accomplished via the low-level I/O `iofunc` library. The outer application-software level provides an interactive user interface.

The three-layer structure of the software permits maximum isolation between modules. In fact, in the early stages of the software development, the kernel functions were tested without the actual hardware. `iofunc` was replaced with an equivalent library that redirected all output to a PC console and acquired input from a test file.

The logic-analyzer hardware operates in two modes—Program and Measurement. The operating mode is determined by the logic level of analyzer's BUSY line.

After powerup, `initall()` puts the hardware in Program mode. The initialization phase includes programming the 8255 chip. `modein()` and `modeout()` are used to reprogram and refresh ports A and C of the PPI. Port B of the 8255 is set up as output for accessing the analyzer's internal registers and set as input for reading the RAM contents.

While reprogramming the 8255 chip, bytes stored in I/O ports can be lost. Therefore, each time the chip is reprogrammed, the values stored in all ports have to be refreshed.

After initialization, the measurement parameters (i.e., trigger pattern, clock source and frequency, active clock slope, and gate level) have to be set up.

The trigger pattern is set up as a string of 0, 1, or X with the leftmost character denoting the probe number 15. The entered string is converted to a 32-bit (for 16 probes) sequence written to the comparator's trigger pattern register.

The data stored in the analyzer's internal RAM can be transferred to the PC by `readbuffer()`. While calling this function, a pointer to the receiving buffer (2048 ×16 bytes) has to be passed.

After transfer, offset 0 in the buffer points to the first vector after the last captured data. `readbuffer()` can be called only when the analyzer operates in Program mode and `modein()` has been called first. After all measurement parameters have been set up, `startcycle()` toggles the START line in the Command register, putting the analyzer in the Measurement mode.

While in Measurement, application-level software can poll the status of

Listing 2—*kernel.h provides basic control of the logic-analyzer hardware.*

```
#define MAXCHAN 16

typedef enum {WR294,RDNEXT,ECWR,HIWR,
              LOWR,WREG,STOP,START}          T_strobe;
typedef enum {FALLING=0,RISING=4}            T_slope;
typedef enum {LOW=0,HIGH=8}                  T_gate;
typedef enum {EXT,F1,F2,F3,F4,F5,F6,F7,F8,
              F9,F10,F11,F12,F13,F14,F15,F16} T_freq;
typedef enum {BANK0,BANK1,BANK2,BANK3,BANK4} T_bank;

void modein(void);
void modeout(void);
void strobe(T_strobe line);
void selectbank(T_bank bank);
void selectslope(T_slope slope);
void selectgate(T_gate gate);
void selectfreq(T_freq freq);
void settrigerpattern(char *pattern);
void seteventcounter(int count);
void setdelaycounter(int count);
void startcycle(void);
void stopcycle(void);
void readbuffer(char *buffer);
int isselected(void);
int isbusy(void);
int rdgate(void);
void initall(void);
```

the analyzer by calling isbusy() rdgate() reports the current logic level present on input line GATE, while isselected() can check whether the analyzer is connected to the PC.

strobe() and selectbank() are used internally by the remaining kernel functions. Listing 3 sets all the programming parameters and starts the measurement cycle.

The measurement ends when either the analyzer is not BUSY or a key is pressed. Following measurement, data is downloaded to the storage buffer and displayed in binary format.

The real application software should provide a graphics interface as well as the ability to display and process data in various formats (e.g., binary, ASCII, assembler listing, or timing waveform representation). For more advanced purposes, data can be stored to and retrieved from files, several runs can be compared and printed, and so forth.

The DOS executable version accompanying the project presents the data either in binary or timing waveform format. The measurement and its parameters can be saved to a file.

The software can also operate in Demo mode. Demo has all the functionality of the real system but doesn't require the logic analyzer's hardware. The captured data is software generated by a random number generator.

Listing 3—*This very simple program shows the sequence of commands needed to operate the logic analyzer. The captured buffer is shown here in binary form.*

```
#include <conio.h>
#include "kernel.h"
char bufferC40961;
main0 {
  int i,c;
  long mask;
/* enter Program mode, set all parameters */
  initall();
  settriggerpattern("XXXXXXXXXXXXX1X0');
  seteventcounter(1);
  setdelaycounter(1);
  selectfreq(EXT);
  selectgate(LOW);
  selectslope(FALLING);
/* enter Measurement mode, wait till end or keypressed */
  startcycle();
  do{
    if(kbhit()) break: }
  while(isbusy());
  stopcycle();
/* back in Program mode, transfer and display RAM data */
  modein();
  readbuffer(&buffer);
  for(i=0;i<4096;i+=2){
    mask=0x10000;
    c=(buffer[i]<<8)+buffer[i+1];
    while(mask>1)
      printf("%d",(c&(mask>>=1))==0?0:1)
    printf("\n"); } }
```

WHAT NEXT?

You probably know the feeling when, after assembling some electronics, you conclude, "I wish I'd designed it the other way."

Bearing this in mind, the most important signals controlling the analyzer are available on two expansion connectors. On J4, the internal data bus (8255 Port B), bank selects, BORROW, and WREG signals are available.

J2 carries RAM address, WR374, WRAM, DIS, and CARRY signals. Additional circuitry can be added on a piggyback board attached to those two connectors.

Initially, the expansion sockets were meant for extra logic probes, but it soon appeared feasible to design different diagnostic and prototyping devices. The piggyback module can support additional probes and banks of RAM.

The expansion board can expand circuitry from the main board-the input latch, comparator with the output multiplexer, and RAM. Signals BORROW and CARRY enable the extension of the comparator and trigger pattern register. Control lines BANK4.. BANK2 can be used to select additional *24* probes, giving a total of 40 channels.

The analyzer's main board has almost all the logic required in a simple digital oscilloscope. After all, it doesn't matter what the data in the analyzer's buffer means. It can represent either multiple of eight binary signals or multiple of 8-bit ADC samples.

For a single analog channel, you need a fast ADC and some glue logic. With the sampling rate of 40 MHz, the bandwidth of the oscilloscope would be 20 MHz.

The hardware can accommodate three analog channels. RAM on the expansion module stores data and can even generate up to 2048 data vectors, thus permitting the use of the analyzer as a simple sequential state machine.

If any of this additional hardware is designed, new software can be added to the kernel.

Having designed and built my logic analyzer, I have to put it into a real test. But for a test, I need to think of another project... ❑

Janusz Mlodzianowski received his doctorate from the University of Strathclyde, Glasgow, Scotland, and is currently a lecturer of informatics courses in the Dept. of Experimental Physics in The University of Gdańsk, Poland. His main areas of interest include microprocessor hardware design, systern programming, and the use of computers in education. You may reach Janusz at fizjm@univ.gda.pl.

SOFTWARE

Complete source code and project schematics are available on the Circuit Cellar Web site. The GAL, kernel listings, and an executable file containing a beta version of a graphics interface are also available at www.bg.univ.gda.pl/-janusz.

I R S

407 Very Useful
408 Moderately Useful
409 Not Useful

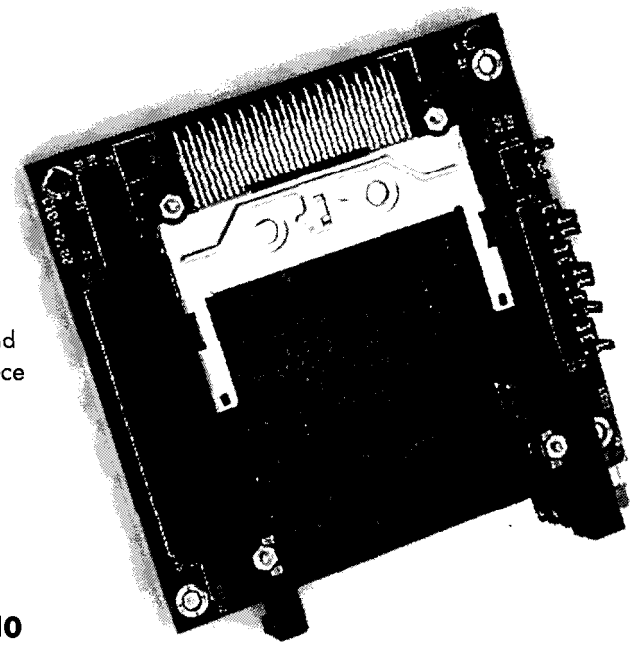
Photo courtesy of
National Semiconductor

NPC

PCMCIA ADAPTER

The **PCM-31** 15 PCMCIA adapter provides PC/104 systems with two independent PCMCIA-compatible slots (accepts separate drivers) and the ability to boot from an SRAM disk. The adapter supports Type I memory cards (e.g., flash, SRAM, etc.), Type II I/O cards [e.g., modem, LAN, etc.), and Type III cards (e.g., ATA mass storage). It is also compatible with Microsoft's FFS-2 flash file system.

The board conforms to the PC/104-standard size of 3.55" x 3.775" and draws 70 mA typical at +5 VDC. The PCM3 1 15 sells for $157 in 100-piece quantities.



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

#510

SOFTWARE-DEVELOPMENT TOOL

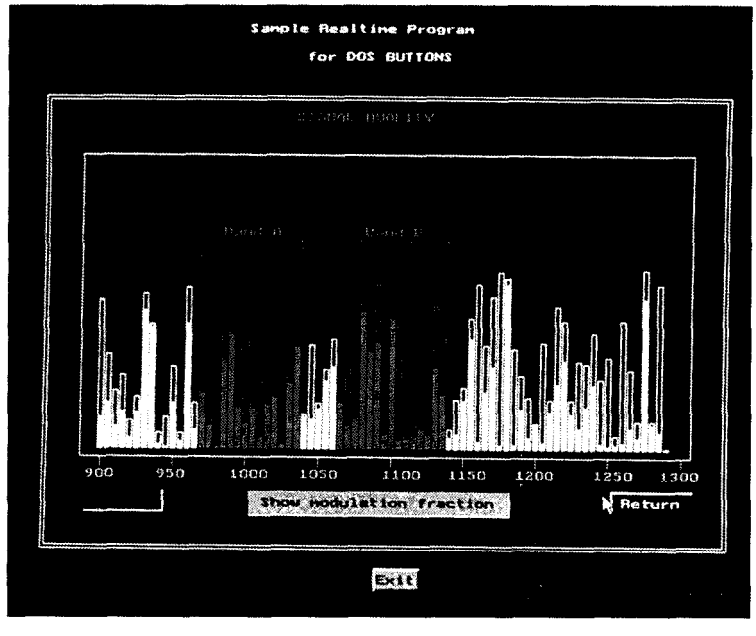
DOS Buttons is a low-cost software-development tool for DOS applications that makes it easy to build simple but powerful GUIs that include windows, buttons, and display bars. Programmers can quickly build memory-efficient, intuitive window-oriented GUIs that accept keyboard, mouse, touchscreen, and pen input. The software is ideal for embedded applications including hand-held data collection, field service and personal communications devices, factory instrumentation, and process-control panels.

DOS Buttons is fully integrated with Communication Intelligence Corp.'s PenDOS, a pen computing environment for DOS platforms that makes it easy to develop new pen-centric applications and run existing mouse-aware DOS applications using pen input. Support for PenDOS enables a DOS Buttons GUI to present pen users with a full keyboard. It also enables the DOS GUI to accept and recognize handwritten input and to capture and compress such input for later use.

DOS Buttons is optimized for embedded environments with tight memory constraints, occupying only 50 KB of memory in the run-time environment. The software is written in C++ and supplied as source code. Software developers access DOS Buttons facilities by making subroutine calls from within their C++ (or other high-level language) programs.

Files containing DOS Buttons images and descriptions can be stored in a text or binary format. The text format enables DOS Buttons GUIs to be altered in the field without recompilation. The binary format provides a 30% reduction in size and enhances security. A utility to convert between the two formats is included.

The production **royalty** for DOS Buttons is $1 per copy with a minimum of 100 licenses.

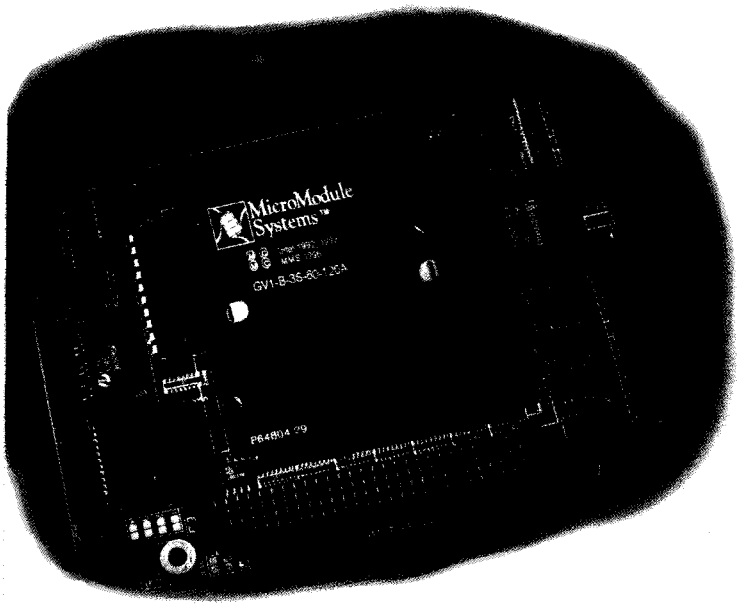


Annasoft Systems
11838 **Bernardo** Plaza Ct.
San Diego, CA 92128
(619) 673-0870
Fax: (619) 673-1432
www.annasoft.com

#511

Nouveau PC

edited by Harv Weiner



PENTIUM-BASED **PC/104-*PLUS*** MODULE

The **ScreamN/104** is a PC/I 04-*Plus*-compatible card that accommodates a MicroModule Systems Gemini CPU module containing a 120-,133-, or 150-MHz Pentium processorwith a PCI chipset, temperature sensor, and 256-KB burst-mode SRAM L2 cache. The board features all the elements necessary to implement a full system on a board (with the exception of video). System components include IDE and floppy drive interfaces, COM1 and COM2 serial ports, parallel port, keyboard, and mouse ports.

The ScreamN/104 features a 64-bit DRAM interface synchronized at twice the PCI clock rate and support for up to 128 MB of memory. It also features a 3.3- or 5.5-V PCI interface for speeds up to 33 MHz.

Pricing for the ScreamN/104 starts at $1200.

parvus Corp.
396 W. Ironwood Dr.
Salt Lake City, UT 84115
(801) 483-I 533
Fax: (801) **483-** 1523
www.parvus.com

#512

DOS-BASED EMBEDDED PC

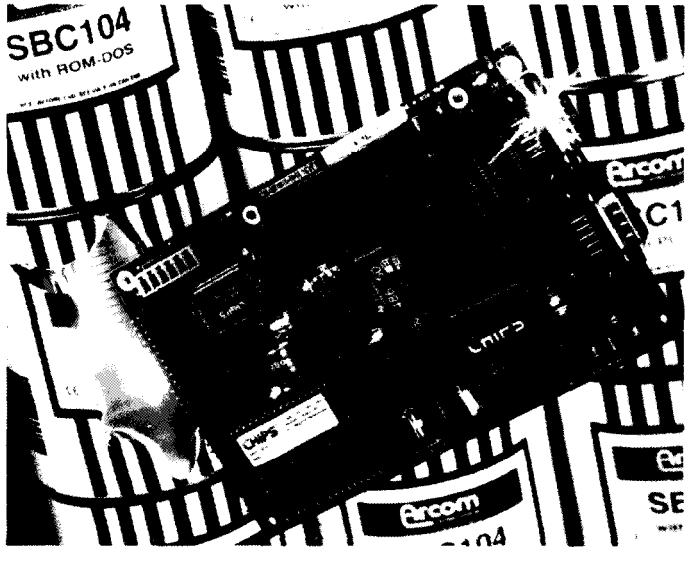
The SBC104 is a DOS-based embedded PC suitable for control applications, networks, or machinery. It isavailablewith an entry-level 25-MHz Intel '386SX CPU or a Texas Instruments 486SXLC2 running at 50 MHz for more demanding applications. Both products are available with either 2 or 4 MB of onboard DRAM.

The SBC104 is supplied with a fully licensed ROM-DOS. Loaded into flash memory and deploying Arcom's Flash Filing System (AFFS), the system builder has a complete DOS-based CPU product without requiring hard or floppy disk drives. In addition to each SBC 104 is supplied with a driver enabling the optional SRAM to be used as a high-speed read/write drive.

Expansion of the CPU board is accomplished by means of the PC/104 interface. Other features include a system watchdog, battery-backed real-time clock, and floppy- and hard-disk drive interfaces via onboard headers. It also includes standard RS-232 COM 1 and 2 via a pair of nine-pin D-type connectors and an AT keyboard connector. A utility disk, supplied with each board, includes a comparative review of ROM-DOS 6.22 and MS-DOS 6.22 as well as a list of comparative commands.

Prices for the SBC 104 range from $295 for the '386SX with 2-MB DRAM and l-MB flash memory to $399 for the 386SX and $450 for the 486SXLC, both with 4-MB DRAM and 2-MB flash memory. Extended temperature ranges are also available.

Arcom Control Systems
13510 **S.** Oak St.
Kansas City, MO 64145
(816) 941-7025, x254
Fax: (8 16) 941-7807
www.arcomcontrols.com **#513**



Nouveau PC

NPC

DIGITAL OUTPUT BOARD

The IDOxx Series is a family of plug-in boards providing one, two, or three groups of 16 optoisolated digital output lines. The Series is particularly useful in factory-floor applications in which large numbers of external relays need to be controlled and where the surge protection afforded the host PC by channel-to-channel and channel-to-computer isolation is desirable.

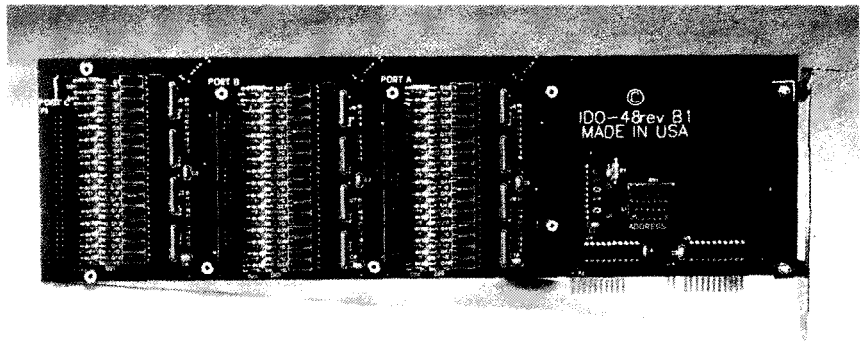
Solid-state P-channel FET switches are used as the output elements to provide greater reliability and much faster turn-on (50 ms) and turn-off (2 ms) times than is possible with electromechanical relays. Output connections are via 50-wire ribbon cables that mate with headers on the board.

The boards provide safety features for up to 48 parallel, differential input voltages from 5 to 60 VDC, including an onboard shield that prevents the user from coming into contact with high input voltages. Isolation of 500 Vrms is provided between channels and also between each channel and the host PC to further protect users from accidentally contacting high voltages. All analog outputs remain disabled until written to prevent spurious outputs from causing damage at system powerup or reset.

The Series comprises three models-IDO16, ID032, and ID048 (16, 32, and 48 channels, respectively). All boards occupy a full-length AT slot and come with a user manual and utility software on disk.

Pricing for the IDOxx Series starts at **$159.**

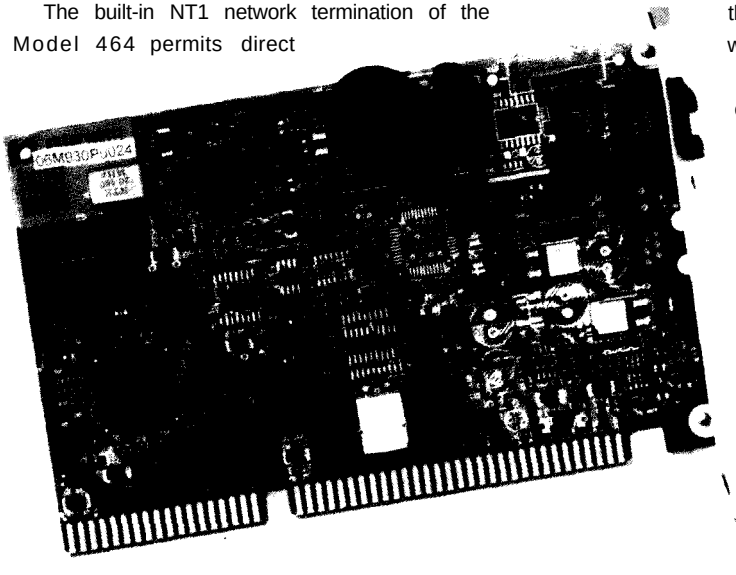
Industrial Computer Source
6260 Sequence Dr.
San Diego, CA 92121
(6 19) 677-0877
Fax: (6 19) 677-06 15
www.indcompsrc.com **#514**



ISDN TERMINAL ADAPTER

Telebyte Technology has introduced a low-cost ISDN terminal adapter for internal PC applications. The Model 464 provides 64- or 128-kbps communications links for data applications between users and remote locations.

The built-in NT1 network termination of the Model 464 permits direct connection to the ISDN line, via the U interface, without additional hardware. The adapter card includes a 16-bit ISA-bus interface.



The Model 464 complies with the switching protocol of AT&T-5ESS, Northern Telecom DMS-100, and National ISDN-1/2. For the international market, the Model 464INT provides compatibility with Euro-ISDN and Japanese INS-64.

The software supplied with the Model 464 includes a quick and easy Windows 95 plug-n-play installation. Additional software includes the Microsoft ISDN Accelerator Pack for Windows 95.

The Model 464 sells for $99.

Telebyte Technology, Inc.
270 Pulaski Rd.
Greenlawn, NY 11740
(5 16) 423-3232
Fax: (516) 385-8184
telebytetechnology.com **#515**

NouveauPC

CIRCUIT CELLAR INK

EMBEDDED PC DESIGN CONTEST WINNERS

For the first time, *Circuit Cellar INK* joined with industry leaders to bring you a bigger, better contest—one with $10,750 in cash prizes.

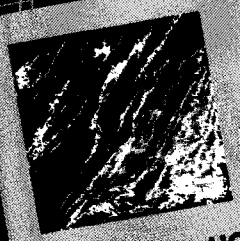
The problem, however, proved to be making a PC-based design contest that didn't cost you a fortune in CPU boards, I/O boards, RTOSs...never mind the time that programming such a complex system would involve.

Our solution: engineering specs. Convince our judges that your product is marketable, efficient, unique, technically viable, and practical. The only hitch—contestants had to make the best selection of products for their application from the participating sponsors.

The sponsors, of course, included a wide range of companies: Ampro Computers, Datalight, National Semiconductor, Paradigm Systems, Phar Lap Software, QNX, RadiSys, Analogic, Grammar Engine, Megatel, M-Systems, The Saelig Company, Teknor Industrial Computers, Toronto MicroElectronics, Tri-M Systems, and VenturCom.

And you came through in splendid form. You brought us an array of submissions, each involving a unique application of embedded-PC technology. Two ambitious readers even submitted three designs each! My hat is off to you. Of course, the entrants' challenge now is to turn their specs into viable products.

—JANICE HUGHES



1ST PLACE

—MARK ROBERTS

WEB-BASED HOME CONTROL SYSTEM

The commonality of the Web has introduced a sophisticated, universal graphical user interface into current technology. Even modest controllers have access to the HTTP and HTML protocols. With such advances, it's not hard to imagine people calling home to receive telephone messages and to "browse" their house. Mark shows how to combine a low-cost, serial-port-driven X-10 home control system with National's NS486SXF and Phar Lap's ETS Web Server to gain Web-browsing capabilities.

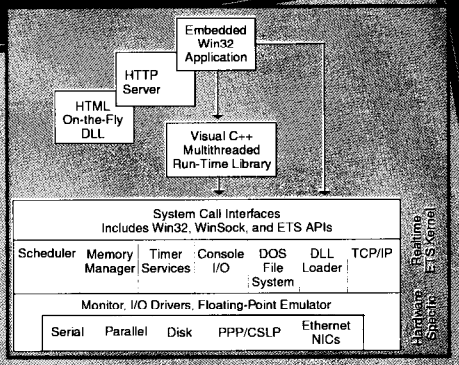
The NS486 is an ideal platform as it provides adequate performance, ample memory, serial- and parallel-port peripherals, and multiple ways to add an Internet connection. The memory shipped with the evaluation board can be easily upgraded to 4 MB, and the onboard flash can be augmented via a PC Card or PC/104 memory board. Alternatively, the firmware can be downloaded into RAM and run out of memory as long as there is no power interruption.

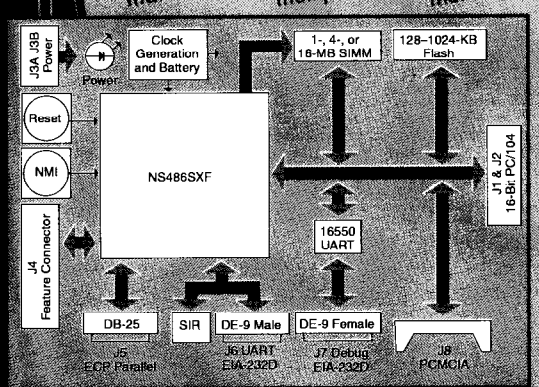
Whether the telephone line is ISP or POTS determines what type of network connection is used. If Ethernet is required, the networking board should be selected to be compatible with the prewritten driver from Phar Lap (i.e., an NE2000-compatible PC Card or PC/104 solution). Since the number of bytes in HTML pages tends to be small, a modem would adequately cover the network connection.

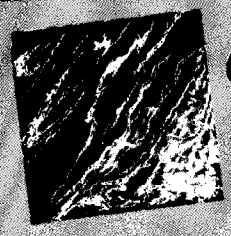
The CM11A offers a sophisticated X-10 interface that is battery backed and capable of storing macros and waking up a host computer with a Ring Indicate interrupt. The unit is programmed via the PC COM-port connection. The CM11A signals the host computer whenever it starts to receive X-10 data.

Phar Lap's Embedded ToolSuite and its MicroWeb server provide a complete development system and environment. The MicroWeb server provides server-side HTML generation. Fields in a standard, precreated HTML document can be tagged to contain real-time data for presentation to the user when the page is accessed. The data can be simple alphanumeric results, status information, or the more complicated graphics from a Connectix camera.

In addition to the $5000 first prize, Mark received the TNT Embedded ToolSuite Realtime Edition, V.9 from Phar Lap Software. You may reach him at mark.roberts@nsc.com.

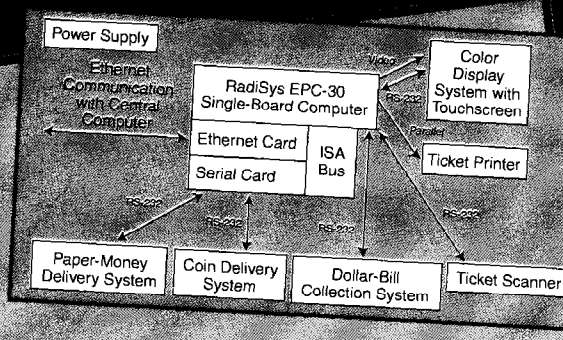






2ND PLACE

—BRAD REED



ATM-STYLE RACE-TRACK WAGER KIOSK

Since the consumer has come to readily accept the ATM and lottery-ticket vending machines, Brad is willing to hedge his bets that a race-track wager kiosk would also be a Derby-style winner.

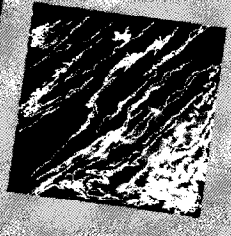
He proposes a full-color monitor with snappy graphics picturing the selected animal and listing the odds at the time of the bet. Using a touchscreen, the wager could be placed on the race, type and amount of wager, and the animal. The gambler would feed the appropriate amount of money into the dollar machine, and a ticket would be printed and returned to the person placing the bet.

Cash payouts could be collected by feeding the winning ticket into the ticket-reading machine. After verifying winnings, the machine would pay out the appropriate amount of paper money and coins. Large winning tickets could be directed to a ticket window for payout.

Brad intends to anchor the electronics of this project around the RadiSys EPC-30 single-board computer. The EPC-30 provides a complete '386 PC-compatible system running at 33 MHz. This system would control a 12" monitor, dollar-bill machine, coin and paper-money delivery system, and ticket reader via serial ports. The ticket printer would be attached to the EPC-30's parallel port. A modem would take care of communications with the race track's main computer.

Because of its ability to handle serial and Ethernet connections and its full-featured GUI, the QNX MicroGUI is the best software option. From a tiny microkernel (32 KB), it manages the interprocess communication between the various system tasks. It uses an MMU to manage processes, making it quite robust even in the fast-changing real-time environment of the race track.

As a second-place winner, Brad received $3000. You may reach him at bradreed@ix.netcom.com.



3RD PLACE

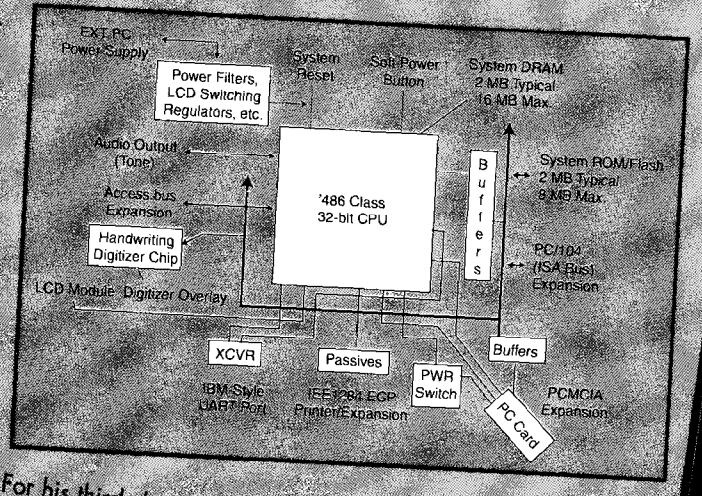
—MARK ROBERTS

AN NS486SXF-BASED PLATFORM FOR NEUTRINO AND PHOTON

Many applications require a standard architecture complemented by a GUI, flexible peripheral expansion, and a general-purpose operating system. However, pulling all these items together can be an expensive proposition that gets even more costly depending on the form factor, solid-state file system, nonstandard display, and input devices. Mark's goal: create a high-quality design with scalable hardware and software.

The only way to provide a comprehensive and affordable product is to use the "system-on-a-chip" processor solution of National's NS486SXF. It guarantees PC compatibility, minimizes BIOS and DOS-licensing expenses, and has the hooks for whatever peripherals are needed for the specific design.

By using QNX's Neutrino/Photon OS, the customer can scale down to 32 KB or up to a full-blown networked distributed system necessary to run something like a whole plant. Networking between Neutrino machines is inherent—you don't need a TCP/IP stack on your local machine. The GUI is easily adapted to any display and input device, and the look and feel of the display can be easily customized by the application programmer. QNX's application builder utility speeds time to market.



For his third-place finish, Mark won $2000. You may reach him at mark.roberts@nsc.com.

HONORABLE MENTIONS

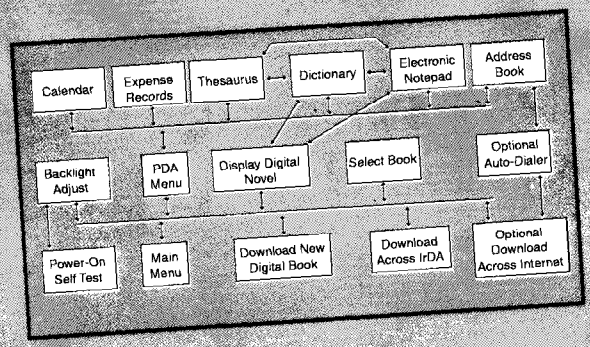
PORTABLE DIGITAL BOOK

—BRAD REED

Plagued by hours of plane travel and nothing to do because the lighting is so bad?

If so, Brad has the solution for you. Using National's NS486 and QNX's Photon MicroGUI, he designs a digital book that is lightweight and battery operated and has two page-size backlit LCDs. Using a touchscreen interface, irDA interface, and optional modem, you can download the latest digital novel from the digital kiosk at the bookstore or purchase it off the Internet. Other applications such as E-mail, address book, and PDA software round out the Digital Book's utility.

As an Honorable Mention winner, Brad received $250. You may reach him at bradreed@ix.netcom.com.

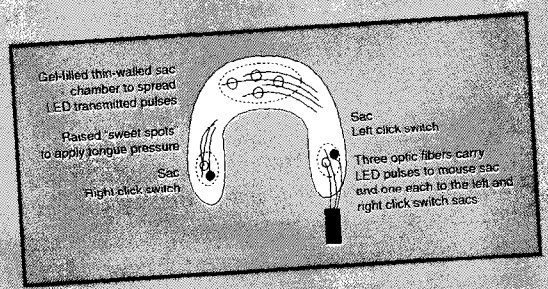


ORAL MOUSE AND GPS WHEELCHAIR

—KELVIN MOK

Severely handicapped people have a difficult time achieving any kind of independence. Kelvin therefore proposes an oral mouse in a fiber bundle to give feedback to a personal computer, which in turn guides a motorized wheel chair or repositions an articulated bed. The other tubes in the bundle would include drink, liquid nutrition, medication dosages, and liquid drainage tubes. Kelvin coordinates the system using an oral mouse; Tri-M's AAEON SBC-670, AAEON PCM-3521 flat-panel CRT VGA module, and ANDI-servo dual servo motor control; M-System flash memory; and Datalight's ROM-DOS software.

Kelvin won $250 and an additional $100 from Tri-M Systems for his design. You may reach Kelvin at pmok@ee.ualberta.ca.

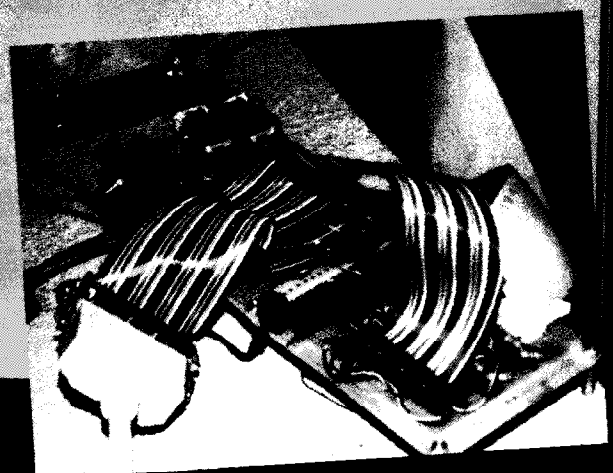


USING AN EMBEDDED PC TO CONTROL INDUSTRIAL EMBROIDERY MACHINES

—VANESSA BINGGELI-COX

Modern industrial embroidery machines are expensive, especially given that the older machines were so well built that they still function just fine. The solution: develop custom-design software capable of translating scanned or clip-art designs into instructions that can drive old industrial embroidery machines. Custom C++ software takes HPGL files created by Corel Draw. These files give the necessary instructions to direct the stepper motors, main machine motor, and any required belt and pulley changes. Coordinating the entire system is Datalight's ROM-DOS along with Megatel's Small PC.

As an Honorable Mention winner, Vanessa received $250 as well as a BD-PC/II+i OEM kit from Megatel.



If you'd like more information about a project, you may contact its designer via E-mail. If you don't have E-mail access, we'll forward your letter to the designer. Just send it care of: Design Contest Winner Circuit Cellar INK • 4 Park St. • Vernon, CT 06066

EPC

Francis Deck

High-Speed Logic Analyzer for Windows 95

While if's relatively simple *to* directly read logic signals using *the parallel* port of a PC, if's sluggish and timing is not reliable. Instead, Francis builds a small *FIFO* circuit *that* can sample *at* over *20 MHz* and is clock independent.

A variety of useful test instruments are available for diagnosing embedded systems. One of the most important-a logic analyzer-can be used to observe relationships between the multiple signals generated byfirmwareorhigh-speedcommunications.

The simplest logic analyzer I can think of is constructed by directly reading logic signals via the parallel printer port of an IBM-PC-compatible computer. Yet, such an analyzer is doomed to be relatively slow, and its timing strongly depends on the clock speed and interrupt structure of the PC.

Last year, I presented a digital sampling oscilloscope (DSO) circuit for the Macintosh ("A Simple DSO Circuit for the Macintosh," *INK* 70). Using the same basic circuit idea, I'vebuiltan inexpensiveeight-channel logic-analyzer circuit that connects to the parallel printer port of a PC-compatible system.

The new circuit uses a FIFO memory chip as a fast data cache and offers variable sampling rates up to 20 MHz. With extremecare in circuit layout, construction, and component selection, rates of up to 80 MHz should be feasible, comparing favorably with DMA transfers on PCI-bus computer systems.

As an added benefit, the sampling rate is unaffected by the timing of the PC. And, the circuit can operate even on PC systems with clock speeds much lower than the sampling rate.

The 20-MHz maximum sampling rate I chose **for** my breadboard prototype-though relatively sluggish compared to a fast desktop computer-is adequate for many embedded applications. For instance, the popular 805 1 and PIC microcontroller families have maximum instruction rates of I-5 MHz. Thus, the logic-analyzer circuitcan serve **as** a useful debugging tool for MCU firmware.

Total parts cost is around $50, and all components are industry standard. The all-CMOS circuit is forgiving enough to work with no problems on a breadboard. Its low power consumption makes it ideal for battery operation.

The Mac DSO circuit used a PIC microcontroller to handle timing and serial communications. For the logic analyzer, all the software resides on the PC, vastly simplifying both construction and programming.

I originally wrote a program to control the logic analyzer in Microsoft QBasic for MS-DOS, while wondering how to deal with the printer ports in Windows 95. But, Craig Pataky's LPTCON device-driver software ("Getting Beyond the Box With Windows 95," *INK* 74) showed me how to create a 32-bit application program.

Because Craig already covers how to use C to program the printer ports, I chose Visual Basic for writing my support software. Thus, code modules for manipulating the printer ports are now available.

By the time you read this, I hope to have BASIC and C versions of the logic-analyzer software for you to download. For existing MS-DOS systems, I've included the original QBasic program, too.

CIRCUIT DESCRIPTION

The schematic of the logic-analyzer circuit is shown in Figure 1. The brains of the

Size (KB)	2	4	a	16	32
Sharp Microelectronics	LH540203	LH540204	LH540205		
Advanced Micro Devices	AM7203	AM7204	AM7205		
Integrated Device Technology	I DT7203	IDT7204	IDT7205	IDT7206	IDT7207
Cypress Semiconductor	CY7C429	CY7C433			

Table *1—The* FIFO chip is now an established industry standard, and devices with capacities ranging from 256 bytes to 32 ***KB*** are readily available. Here, chips with 2 *KB* ***or*** *more are* listed.

circuit is U5, an industry-standard FIFO memory chip.

The Dallas Semiconductor DS20 13 FIFO chip specified in the Mac DSO article was discontinued, but several other manufacturers turned up when I searched the Web. A list of compatible parts is provided in Table 1.

Capacities ranging from 256 bytes to 32 KB are offered with access times as fast as 12 ns, suggesting that sampling rates approaching 80 MHz are possible with careful circuit layout and construction.

Larger chip capacities are doubtlessly better for serious analysis, but 2 KB is more than adequate for routine work. For instance, 2 KB is enough to monitor MCU programs hundreds of bytes long and fill up the screen with data many times over.

Effective use of larger capacities requires writing custom analysis software to automate the search for specific combinations or sequences of logic states. Nonetheless, the support software is easy to modify.

Theeightlogic inputsare buffered byU1, a74HC573 octal latch that's permanently tied open. Using this chip means the circuit is designed for 5-V signal levels, but you can substituteanother bufferchiptosuitthe logic family you'reworking on. For routinework, the exact matching of logic families is prob ably unimportant. Yet, the buffer protects the expensive (upwards of $30) FIFO chip.

A 20-MHz clock oscillator at U 1 generates the timebase for the circuit. Several lower frequencies are provided by U2, a 74HC4040 counter. The PC can select one of seven arbitrarily chosen frequencies or an external clock input using U3, a 74HC15 1 data-selector chip.

The PC can also disable the *SCLKclock signal altogether. An inverted clock signal, SCLK, is available, and one of these two signals might be useful for driving an external data-acquisition chip such as the flash ADC used in the original Mac DSO circuit.

One of the inputs to U2 serves as an external clock input that can be driven by the clock of the system you're testing. Then, each data byte in the FIFO corresponds to a single clock cycle in the circuit under test.

A reset signal is also provided and can be used to drive the reset line on an MCU chip, though you should probably buffer this line to protect your PC. When the PC releases the reset signal, your firmware and the FIFO start up at the same time, and you can monitor the progress of your program through hundreds of instructions.

Successive *SCLK pulses at the *W line of the FIFO cause data bytes to be written into memory. The FIFO incorporates its own address counters and read/write circuitry, substantially reducing the parts count. To erase the FIFO chip, the *SCLK signal must be in a high state.

For widespread compatibility, the circuit design assumes the printer-port lines are unidirectional, as was the case in the earliest PC systems. The Data Output and Status Input lines used by the circuit are labelled DO-D6 and S3–S7, respectively.

There are only five handshake inputs, so data bytes are read by the PC as pairs of four-bit nybbles through U6, a 74HC157 four-of-eight data selector. One of the data lines is inverted in the standard PC parallel port-a problem corrected by a software XOR operation during download.

Three additional lines on the FIFO are interfaced to the PC. The *FF (full flag) signal indicates the FIFO is full and has stopped accepting write operations. The *R [read) line retrieves a byte of data from the FIFO and places it on the output lines of the chip.

By applying successive *R pulses and reading successive nybbles through U6, the' entire FIFO contents can be quickly downloaded to the PC. The chip can be erased by placing a low signal on the *RS (reset) line when the *W signal is in a high state.

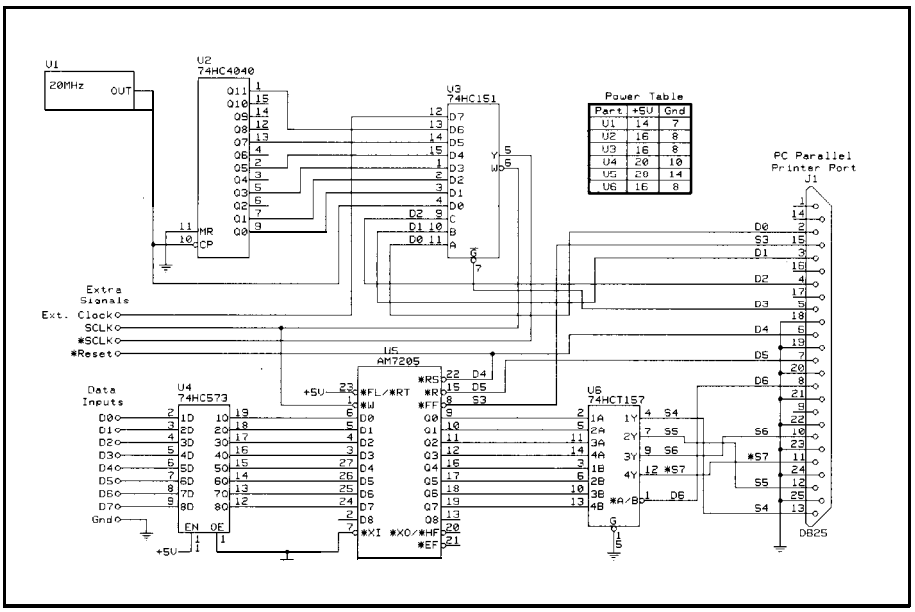


Figure I-A standard FIFO memory chip and some common CMOS logic are all that's needed to build a high-speed logic-analyzer circuit.

CONSTRUCTION

Many of the traditional reasons for avoiding breadboards stem from the asymmetric I/O characteristics and high current consumption of TTL logic. My breadboard prototype worked just fine, thanks to the exclusive use of CMOS circuitry.

Nowadays, I seldom commit my small projects tocircuit boards, since breadboard strips are quite cheap. Besides, the time it takes to design a board always seems to be better spent improving the software.

Power consumption during idle turns out to be less than 10 mA, so battery operation is possible. The cabling to the PC should be no longer than a couple feet, and the ground wire should be at least 18 gauge.

If you use a ribbon cable, hook up all eight ground lines (pins 18-25) of the printer port to minimize the ground resis-

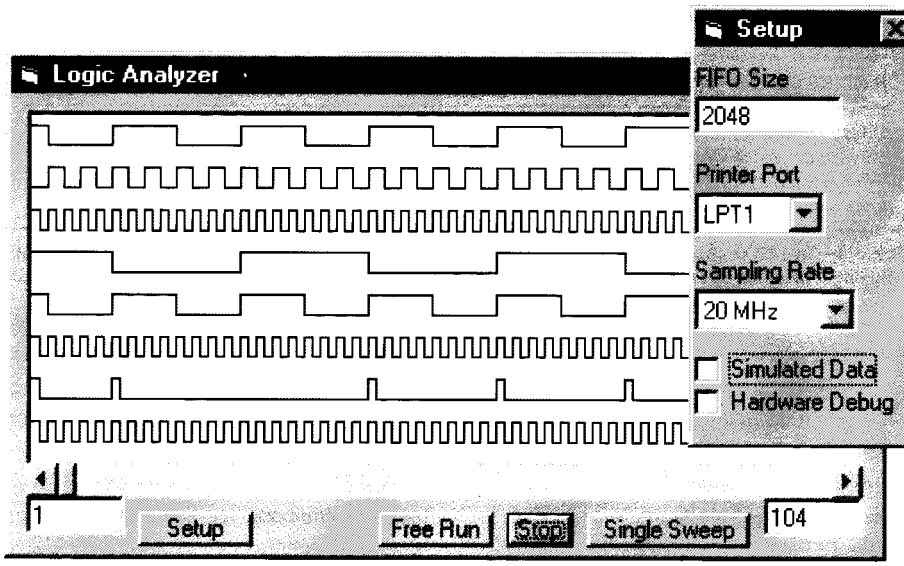


photo I-Visual Basic is ideal for designing custom user interfaces. The *L PTCON* device *drive* provides full access to the ***printer-port*** hardware.

tance. Also, I suggestusing a heavy ground bus on the circuit itself, with copious decoupling capacitors.

Interestingly, the circuit seems to work when the power supply isn't hooked up at all. **The** decoupling capacitors arecharged through the protection diodes of the CMOS chips, some of whose inputs are held at a high state.

However, be aware that a low power-supply voltage causes the inputs of U4 to draw current from the circuit you're testing. As another precaution, consider that old printer-port cards may need pull-up resis tors since they used TTL output drivers.

This circuit may be dangerous if exposed to high voltages, and I discourage the use of this circuit in any environmentwhere high voltages are present. I don't trustthegrounc connection of the printer port for safety and the circuit might not be grounded at all if a notebook computer is used.

SOFTWARE OPERATION

Because I like keeping budgets to a minimum, I bought the Learning Edition o

Listing I-For the sake of clarity, the data-acquisition subroutine is shown without variable declarations or diagnostic code. The routine calls o code module that accesses the *L P TCON. VXD virtual* device driver.

```
'collect data for real from analyzer circuit speedily
Sub collect(port As Long, sampRate As Integer, fifolen As Integer)
  'define I/O lines and "idle" state
  Const fclk = 1                    'clock frequency select bits
  Const clken = 8                   'clock enable, *E line on 74HC151
  Const rstfifo = 16                'reset, *RS on FIFO
  Const rdfifo = 32                 'read, *R on FIFO
  Const ab157 = 64                  'nybble select, SEL on FIFO
  Const fullfifo = 8                'full flag, *FF on FIFO

  idle = sampRate * fclk + clken + rstfifo + rdfifo
  lptcon_writedata port, idle       'reset FIFO
  lptcon_writedata port. idle - rstfifo
  lptcon_writedata port, idle
  lptcon_writedata port, idle - clken 'enable sampling clock
  do                                'wait until FIFO is full
  loop until (lptcon_readstatus(port) and fullfifo) = 0
  lptcon_writedata port, idle       'disable sampling clock
  for i = 1 to fifolen 1            'read data from FIFO byte by byte
    lptcon_writedata port, idle rdfifo
    lo = (lptcon_readstatus(port) Xor 128) \ 16
    lptcon_writedata port, idle rdfifo + ab157
    hi = (lptcon_readstatus(port) Xor 128) And 240
    sigData(i) = lo + hi
    lptcon_writedata port, idle
 next i
End Sub
```

Microsoft Visual Basic 5.0 for personal use. The Professional Edition comes with many more useful bells and whistles and can create ActiveX custom controls as well as fully compiled executables.

Nevertheless, the Learning Edition supports full Windows API access and has proven quite serviceable for general-purpose programming.

The critical data-collection subroutine is shown in Listing 1. The routine starts out by disabling the clock and erasing the FIFO chip. It then re-enables the clock at the desired frequency, causing data collection to begin. Inside an appropriate time-out loop, the routine waits for the *FF line to be pulled low.

During download, the clock isdisabled. Each data byte is then read from the FIFO chip by pulling the *R line low and reading two successive nybbles through the data-selector chip. Data bytes are stored in an integer array for subsequent display or analysis.

The rest of the program is devoted primarily to the user interface and is quite straightforward. A screen shot is shown in Photo 1. A horizontal scroll bar enables you to browse an entire sweep of data. Special analysis features (e.g., searching for a particular pattern) are easy to implement if you don't mind writing a few lines of BASIC.

USE THE TOOLS

The logic-analyzer circuit is a powerful diagnostic tool and provides an important benefit. Because the PC's CPU must process interrupts, accurately timed sampling rates are difficult to achieve with software alone.

This difficulty can be overcome by complicated and expensive DMA circuitry, but the FIFO cache used by the logic analyzer is an inexpensive alternative. The analyzer circuit can be used for uncannily accurate timing measurements. Also, theclockspeed of the PC doesn't affect the sampling rate.

You'll notice I left one of the printer port Data Output lines unused. Also, the outputs of the FIFO are in a high-Z state when a byte is not being read from the chip. Thus, multiple FIFO chips could share a common output bus and control signals.

Tying the *R lines of each FIFO to a separate output from the PC enables the circuit to be expanded to support wider data paths. Of course, a small software modification would be necessary.

If this circuit helps you discover one logic error, firmware bug, or failed chip, then it's worth the trouble of constructing it. The analyzer can be used for a variety of tasks related to communications, including deciphering data protocols, baud rates, and so forth. And, the circuit retains its compatibility with the **ADC** chip used in the Mac DSO circuit.

All in all, it's a versatile yet inexpensive tool for all sorts of design and diagnosis tasks in embedded systems. IPC

Francis J. Deck received his Ph.D. in physics from the *University* of Notre *Dame. His* main technical interests are instrumentation and control engineering, *and* software development. you *may* reach Francis via *members.aol.com/fdeck/main.html.*

SOFTWARE

Source code for this article and Craig Potaky's I PI CON virtual device driver ore available on the Circuit Cellar Web site. You con download the latest version of the software, including source code, from Francis's Web site listed above.

SOURCES

AM7203, AM7204, AM7205
Advanced Micro Devices, Inc.
P.O. Box 3453
Sunnyvale, CA 94088
(408) 732-2400

AMD FIFO **Chips**
Newark Electronics
12880 Hill Crest Rd.
Dallas, TX 75230
(972) 458-2528
Fox: (972) 458-2530
www.newark.com

CY7C429, CY7C433
Cypress Semiconductor
3901 N. First St.
San Jose, CA 95 134
(408) 943.2600
Fax: (408) 943-6848
www.cypress.com

IDT7203, IDT7204, IDT7205, IDT7206, **IDT7207**
Integrated Device Technology
3236 Scott Blvd.
Santa Clara, CA 95054
(408) 727-61 16
Fox: (408) 988.3029

LH540203, LH540204, **LH540205**
Sharp Electronics Corp.
Microelectronics Group
5700 NW Pacific Rim Blvd., Ste. 20
Comas, WA 98607
(206) 834-2500
Fox: (206) 834~8903

IRS

4 10 Very Useful
41 1 Moderately Useful
412 Not Useful

PC/104Quarter

Rick Lehrbaum

PC/104 Gets Tough

Ruggedizing the Embedded PC

Despite what we put up with from our desktop machines, we expect embedded PCs to run like clockwork, regardless of the environment. Rick shows us how fo ruggedize PC systems to meef the tough demands of the real world.

I once asked a Hong Kong desktop-PC motherboard supplier how many design engineers worked at his company.

"We don't need design engineers," he explained. "Each month, we locate the cheapest chipsets and ask our PC-board supplier to send us bare motherboard fabs that match the chipsets we plan to use."

"Don't you need to understand the designs technically or do worst-case timing and bus-loading analysis to be sure the motherboards you build will be reliable?" I asked, incredulous.

"Why bother? Chipset prices are so volatile, we can't afford to invest in that much engineering. Besides," he added, "we know the designs are solid because they're made in the tens of thousands each month by manufacturers here and in Taiwan. Any problems have been found and fixed."

I was amazed. Is this the "We don't need no stinkin' engineers!" strategy of computer manufacturing?

This brief interchange brought home the enormous difference between the priorities of the desktop-PC market versus those of the embedded market. I wondered, how can the components and technologies of the desktop market possibly be made robust enough for embedded systems?

Repeatedly, we hear about the benefits of using embedded PC systems. But, we need to watch out for the adage "Live by the sword, die by the sword."

Sure, your blood pressure goes up a little and you utter a few choice words when Win95 displays its all-too-familiar "Application not responding; terminating task" message after you've spent half an hour writing a report. But if the computerized toll booth you designed lets everybody cross the Bay Bridge free one day due to a system crash, it's another story.

Fact is, we expect embedded systems to run likeclockwork. All day. Everyday. And in some pretty nasty environments. Embedded systems answer to a higher authority—the real world.

So, how can you be sure your PC/I 04-based system meets this higher standard?

HELLISH ENVIRONMENTS

Let's first think about the typical PC/I 04-based embedded-application environment. It could be almost anything, but let's assume it's commercial, medical, industrial, mobile [see Photo 1), or military.

These five markets have embraced PC/I 04 as an alternative to roll-your-own electronics. Although every embedded application is truly unique, Table 1 summarizes some key concerns in these markets,

These constraints are placed on the internal electronics by the system designer. The total packaging-including protective devices like fans, heat sinks, EMI shielding, shock mounts, and so on-almost always supplements the specs of the available embedded electronics to fully meet the needs of the application's external environment.

HELP FROM MY FRIENDS

Fortunately, PC/104 modules are designed for embedded applications. So, you expect them to meet the needs of embedded applications, right?

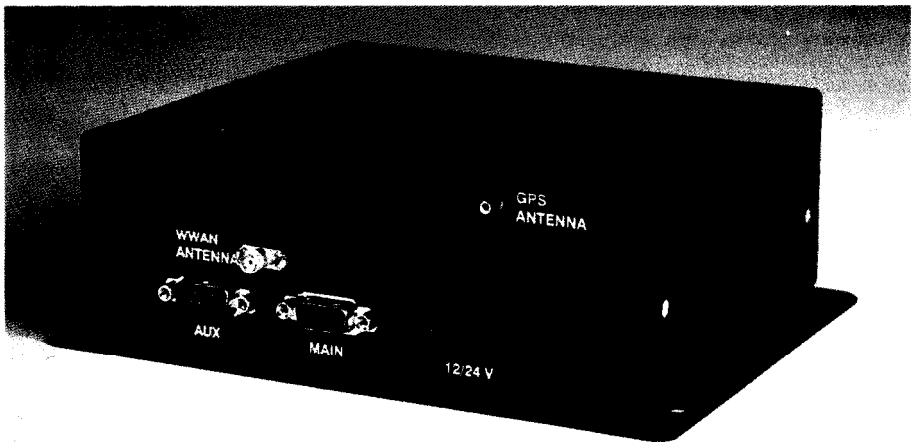


Photo I-The ***PC/104-based TrackRecord*** I from Mobile Integrated Technologies provides wireless communication and GPS position information on a PC-compatible mobile network gateway. It enables trucking companies to track in ***real*** time the location and contents of their fleet. ***Based*** on an internal ***PC/104*** embedded PC, this rugged system meets a long list of SAE environmental standards.

To illustrate how close they come, Table 2 summarizes some of the environmental specs met by Ampro's PC/104 modules. Ampro tests all new products to these specs as part of the design-qualification process.

You're probably wondering how it's possible to consistently meet such specifications, given the desktop/retail origins of PC-compatible architecture, components, and peripherals. Read on.

KEEPIN' COOL

They say the only things you can count on in life are death and taxes. Likewise, there's one trait all electronics share-they generate heat. And, heat is a great impediment to reliability.

First of all, the more heat, the more power is being consumed. Power budgets are always limited in embedded systems—either by the capacity of the system's power supply or by a portable or mobile system's operational battery life.

In a lot of PC/104 applications, the embedded PC replaces the 8051 or 68HC1 1 single-chip micro of a previous design. The allowable power budget may be a couple of watts!

Also, the more heat, the lower the system's MTBF. took at the MIL-HDBK-2 17F standard for determining theoretical MTBF, and you'll see MTBF is inversely related to temperature. If you keep your system's electronics cooler, they last longer.

Thermal expansion and contraction of components stress interconnects throughout the system, including wire bonds within ICs, solder joints on PC boards, and elsewhere.

And, don't forget functional reliability. Electronic designsareonlyasgood as their timing diagrams, and circuit timings vary with temperature. For best operation, keep temperatures under control. System crashes or inaccurate sensor readings can result from excessive heat buildup.

To reduce heat in an embedded application, either reduce power consumption or get rid of the heat generated by your electronics as efficiently as possible. Let's explore both options.

First, resist the temptation to use the fastest CPU available. Maybe all you need to be is a bit more clever (or persistent) in implementing your software.

A '386SX can do a lot of useful work in real-world embedded applications. After all, 16-bit CPUs have only recently become more common than 8- and 4-bit ones in most appliance-like applications.

Those of us old enough to have used CP/M-based WordStar on 8-MHz Z80s enjoyed speeds faster than with Word 97 under Windows 95 on a Pentium.

In many cases, faster CPUs end up running more lines of code per second, not doing tasks morequickly. Fast CPUs and large memories get used up by programmers in no time flat. So to keep your system's power under control, stay away from lightning-fast CPUs.

Displays and disk drives also consume vast amounts of power. The brighter the display, the more power required. Passive LCDs are more miserly than TFT displays, which consume less than Et panels and CRT monitors.

One of the biggest power hogs is that quiet little video controller. Today's high-res GUI-accelerated VGA controllers sometimes consume as many watts as the system CPU. Solid-state disks save you a lot in power, but they cost more per megabyte, except in smaller capacities (2 MB or less).

One advantage of PC/ 104 over conventional PC technology is the reduced requirement (from 24 to 4 mA) for the bus drive. It reduces unnecessary buscurrent, eliminates unneeded bus-driver ICs, and results in lower power consumption. Most PC/I 04 modules consume between 1 and 5 W.

PC/I 04 SBCs are increasingly based on laptop (not desktop) PC chipsets due to the higher integration of functions. So, laptop-PC power-management functions are increasingly available. To take advantage of these power-management hooks, the PC/I 04 system requires BIOS or utility software to activate and control the various power-saving modes.

One laptop-PC standard is Microsoft's Advanced Power Management (APM) spec. With APM, your embedded application can invoke standardized power-saving modes (e.g., sleep, suspend, and doze] in a hardware-independent manner.

In some cases, you can vary the system-clock speed to reduce power when high processing speed isn't needed. Some PC/I 04 SBCs include thermal sensors and software support that automatically throttle down CPU clocks to avoid overheating of components.

	Commercial	Medical	Industrial	Portable & Mobile	Military
Operating Temp. (“C)	0 to +60	0 to +60	-20 to +70	-40 to +a5	-40 to +a5
Shock & Vibration	usually not required	usually not required	usually required	always required	usually required
Humiditv	not reauired	not reauired	not usuallv required-	5-95% (or greater)	5-95% (or greater)
EMI& ESD	FCC Class A	FCC Class B	FCC Class A	varies	varies
Battery Operation	no	no	no	yes	varies

Table I--Different types of applications place varied demands on their electronics.

PCI

Next, get rid of the heat produced inside your system. The main objective is to remove heat from internal hot spots as efficiently as possible.

Heat sinks, which come in many shapes and sizes, eliminate heat by conduction and convection. But, don't expect too much from the heat sink itself. An efficient convection process requires air to move freely within your embedded-system enclosure.

Be sure air can move past the heat sink. You can typically accomplish this by using vents and not impeding the smooth flow of air with cables or other objects.

A case-mounted fan can do wonders, Even a slow, quiet fan dramatically improves heat removal from embedded electronics.

Beware heat sinks with built-in fans—the kind commonly mounted on desktop-PC CPUs. You're introducing a mechanical device-a sourceofeventual failure—within an otherwise solid-state system. It's much easier to diagnose and replace a fan if it's outside the system rather than buried inside and attached to the CPU chip.

CPU heat-sink fans frequently fail shock and vibration tests, with mounting hardware and bearings coming loose or tearing free from their plastic fittings. As well, heat-sink fans have a hard time fitting within the PC/I 04-module top-side height specs.

If you can't have vents, you can still expel heat via conductive heat sinks, heat exchangers, and active cooling elements. Or, you can use unusual heat-removing devices such as circulating liquids, electrically driven activecooling devices (liketiny air conditioners), and piezoelectric "flappers" that flap back and forth to move air but without the inherent weaknesses of ordinary fans.

Photo 2-The Portable Maintenance Access ***Terminal (PMAT)*** from Demo Systems is used on the Boeing 777to access the ***onboard*** aircraft information management system. ***It contains an EBX form-factor SBC and has*** been ***qualified to*** Boeing's ***highly demanding environmental standards.***

Conduction is the mostcommon method of eliminating heat from PC/I 04 electronics in sealed systems. With the heat sink attached to hot ICs atone end and to the system enclosure at the other, heat is piped to the outside environment without fans or vents.

To make good thermal contact between the heat sink and electronics, you can use stick-on (or glued on) heat-conductive gaskets, heat-conductive plastic-foam strips (e.g., Bergquist's Gap Pads or Chomerics' Therm-a-Gap), and liquid-filled plastic bags (e.g., Liquid Heat Sink from Aavid).

How do you know your heat-dissipating techniques are adequate?

It may seem self-evident, but the basic principle is this: no matter what, remove enough heat from the electronics to not violate manufacturer specs. Board makers usually specify maximum ambient operating temperatures, but this information isn't useful unless accompanied by air-flow-requirement specs.

Also consider the maximum IC case temperature. For example, the PGA version of Intel's '486DX CPU is rated to a maximum case temperature of 70°C during operation, whereas the surface-mount tape carrier package (TCP) Pentium Ampro uses for its Pentium-based PC/I 04 CPU module has a case temperature rating to 95°C.

Once you have the information, attach thermal sensors to the hot spots and run your system in an environmental chamber over its intendedexternal temperature range. For maximum reliability, test your system over a wider range than its spec to ensure it works in spite of component variations,

One more thing. Ever wonder how some manufacturers can rate their PC/I 04 modules to operating temperature ranges like -40 to +85°C even though many components (especially PC chipsets) only come in 0-70°C versions?

In varying IC temperature, transistor-switching rates change with temperature. Some switch faster, others slower, depending on process technologies. The net result is that IC signal timingsvaryovertemperature.

In the worst case, a function may completely fail due to race conditions. It's more challenging to make digital systems work over wider temperature ranges, but it requires more conservative designs from the perspective of worst-case signal timing and bus loading. You need to thoroughly wring out the prototypes in a thermal chamber.

Size	3.550" × 3.775" x 0.6"
Weight	2-3.5 oz.*
Power consumption	I-5 W*
Shock	50-G 3-axis peak (per MIL-STD-202F, Method 2138, Table 213-1, Cond. A)
Vibration	11.95-G 3-axis RMS at 100-I 000 Hz (per MIL-STD-202F, Method 214A, Table 214-1, Cond. D)
Operating temperature	0 to +70°C, standard; -40 to +85°C, extended
Storage temperature	-55 to +85°C
Humidity	5–95%, noncondensing
EMI compliance	EN 55022 Class B (radiated & conducted emissions)
EMC & ESD compliance	IEC 801-2 (electrostatic susceptibility) IEC 801-3 (electromagnetic field susceptibility) IEC 801-4 (fast transient susceptibility)
MTBF	ground mobile, at 55°C: 30,000–70,000 h* ground fixed, at 55°C: 150,000–650,000 h* (per MIL-HDBK-217)

* These values vary according to the specific module.

Table 2-Not all ***PC/104*** modules meet these specs. ***Ampro,*** however, has established these environmental requirements for all of its modules.

Voltage also impacts transistor-switching speed. Ampro introduces a high- and low-voltage test at the temperature extremes in a "fourcorners test" (i.e., high temp, high voltage; high temp, low voltage; low temp, high voltage; and low temp, low voltage). We also power the system off at each corner, let it soak while powered down, power it up again, and rerun the full set of functional tests.

However, passing a qualification test suite during prototype development isn't enough. You must repeat the full bank of testing anytime you change an active component-even if it's supposedly an exact replacement.

WET BEHIND THE EARS

What about moisture? What's wrong with a little condensation, anyway? tots!

Moistureon electronics often gets mixed up with chemicals floating in the environment, forming corrosiveacids thateatthose tasty little ICs, resistors, capacitors, and connectors. Obviously, you want to avoid wet electronics.

In most applications, the user is responsible for keeping the system dry and away from overly moist air. But in many applications-especially mobile and portable ones-the laws of physics conspire against the system.

For example, air that seems dry at 70°C starts perspiring when the temperature drops below 0°C. So, unsealed portable or mobile systems are going to be susceptible to condensation on their internal electronics.

The solution? Either seal the enclosure and don't allow exchange of air with the environment, or coat the electronics so those delicious morsels aren't available for the chemicals to lunch on.

Since it's hard to prevent air exchange, electronic assemblies in mobile, portable, and militaryapplicationsareoften sprayed with conformal coatings that protect them from the effects of condensation.

But, those coatings can have undesirable side effects. They can clog connectors and reduce the efficiency of heat dissipation. Be sure to verify that the conformally coated electronics meet your high-end temperature requirement reliably.

And before you go forward with a plan to conformally coat your electronics, you should know that coatings have a bad habit of getting into-and damaging—connector contacts. Consult the manufacturer of your PC/I 04 modules to check the impact on your warranty.

SHAKE, RATTLE, AND ROLL

Your embedded project may need to operate while portable, mobile, or airborne (see Photo 2). And, it must be transported from where it's built to where it will be used. How can you ensure your PC/104-based design survives the shocks and vibrations of UPS delivery-or worse?

PC/I 04 modules are inherently quite stable mechanically for several reasons. The small dimensions (3.6" x 3.8") minimize harmonic oscillations produced by the motions of portable and mobile environments.

The ratio of board thickness to area is greater than for larger board form factors, so the modules are relatively rigid. Also, the four corner mounting holes are spaced closely enough to secure the modules to each other or their enclosure. Finally, thegold-plated pin-and-socket bus connectors provide a large, reliable contact area for signals and power.

An additional, if indirect, advantage of PC/104's small size is that components

are typically surface mounted. If done properly, thissetupcontributesexcellent shock and vibration resistance. In this regard, watch out for crystals and capacitors, which mechanically may represent longish cylinders hung on the board from two relatively weak wires at one end. It's a good idea to use axial, rather than radial, crystals and capacitors whenever possible.

One area of weakness is the typical I/O connectors. They tend to be unshrouded dual-row, 0.1" right-angle headers.

Although they're reliable from an electrical perspective, these connectors don't provide a way to keep mating connectors from coming **off.** In practice, this limitation hasn't been serious becausesystem developers use a number of tricks to overcome the problem.

You can apply a liberal coating of RTV to the junction between the board and mating connectors. Or, make a bracket that holds the I/O connectors in place and attaches to the PC/I 04 module or system enclosure. You can also slip a tie wrap between the two rows of board connector pins and around the mating connector.

Interestingly, this shortcoming hasn't interfered with widespread acceptance of PC/I 04 in portable, mobile, and avionics applications. However, PC/I 04 module designers are increasingly selecting board I/O connectors that offer a way to lock the mating connector in place.

Don't forget your embedded system's disk drives. In mobile and portable applications, rotating-magnetic-media diskdrives need to be shock mounted or-better yet—replaced with solid-state disks. In "To ROM or Not to ROM" (INK 83), I discuss the wide range of available options for doing this in PC/I O&based embedded PCs.

How much shock and vibration must the embedded system withstand? It depends. Good system design goes a long way in protecting the electronics from the shake, rattle, and roil of application environment.

ELECTRIC SHOCK THERAPY

Another area of concern is electrostatic and electromagnetic interference-both generated and received.

With today's PC CPU clock rates commonly in the range of 33–200 MHz, there's a risk of generating HF, VHF, and UHF electromagnetic interference (EMI). One problem might be interference from the embedded PC's internal signals with low-level analog or other sensor inputs.

Never forget-there's no such thing as "digital electronics." It's really all analog. It only looks digital to the naked eye.

Those square-looking waveforms are made up of an infinite number of sine waves. The squarer (cleaner looking) the signal, thegreater the number and strength of the high-frequency components.

There are many ways to reduce EMI input and output. Of course, it's importantto design (or select) system boards by maximizing signal-noise margins and minimizing unnecessarilysharpoutput-signaledges.

If you're debugging your own PC/I 04-module designs, scope out signals to locate ringing and other signal problems that can contribute to excessive radiated EMI. Then, adjust signal terminations and trace routing to clean up the signals and minimize their high-frequency components.

Fortunately, the PC/I 04 standard has some inherent advantages relative to EMI. The reduced bus drive permits the use of low-current bus drivers [e.g. HCT). And,

lower bus current means PC/I 04 modules are lower power radio-signal transmitters.

Another advantage is the modules' small dimensions. They're shorter and therefore less efficient radio-transmission antennas.

Holes in the system's enclosure (i.e., connections to the I/O and power connectors) where signals can leak out (or in) offer the greatest risks. In passing EMI tests, you may need to add shielding or ferrite beads to the cable connectors or wires where the offending signals escape.

Although it can be expensive to modify a module's design, often the best solution is to reroute traces so they don't couple undesirable frequencies from one function line to another. Power and ground plans can minimize radiation (and reception) of RF signals. Tiny surface-mount ferrite beads can also be added directlyon the PC board.

Sometimes, inadequate power-supply bypassing is the culprit. Watch out for the poor-quality tantalum bypass caps or less efficient electrolytic caps often used on products for the clone-PC market.

Remember, electromagneticand electrostatic interference is a two-way street. What transmits efficiently, receives efficiently.

The receive side is known as "susceptibility." You don'twantyour embedded system to fail due to external electromagnetic or electrostatic currents. System crashes, false resets, and damage to electronics can result.

Want to minimize electromagnetic and electrostatic susceptibility? Use the same measures that reduce transmission to reduce reception. In other words, what transmits poorly, receives poorly.

GETTING REAL

Are you feeling hopelessly pessimistic?

The good news is that thousands of embedded systems have been designed, built, and successfully deployed using PC/ 104 embedded PCs in a wide range of environments. What's more, many of these have passed tough FCC, UL, CSA, VDE, FDA, FAA, SAE, and CE-Mark compliance scrutiny.

So, don't despair! All it takes is some careful engineering and a little TLC. EPC DTQ

Rick Lehrbaum cofounded Ampro Computers where he served as VP of engineering from *1983 to 1991. Now, in* addition to his duties us VP of strategic development, Rick chairs the PC/*104* Consortium. He may be reached *at rlehrbaum@ampro.com.*

SOURCES

Alternative heat sinks

Aavid Thermal Technologies
P.O. Box 400
Laconia, NH 03247-0400
(603) 528.3400
Fax: (603) 528-1478
www.aavid.com

Advanced Thermal Solutions
36 Jaconnet St.
Newton, MA 02161
(6 17) 965-8989
Fox: (617) 630-0027
www.qats.com

Bergquist
5300 Edino Industrial Blvd.
Minneapolis, MN 55439
(612) 835-2322
Fox: 16 121 8354156
www. ergquistcompany.com

Chomerics
77 Dragon St.
Woburn, MA 01801
(78 1) 935-4050
Fax: (78 1) 933-43 18
www.chomerics.com

IRS

4 13 Very Useful
414 Moderately Useful
415 Not Useful

APC

Applied PCs

Fred Eady

Interfaces and GUI-Building Packages

Part 2: Emulating Paper Tape

Just because you don't use paper tape any more, don't assume if's out of the picture everywhere. Many old machines still use if, and you never know when you'll have to build a paper-tape emulator. Fred shows you how to do if using em Ware.

You can stop laughing now. I know, I know.. .paper tape-ha! Who uses paper tape anymore?

Well, nobody. Well, *almost* nobody. Paper tape may be ancient, but to many machinists, its memory (and usefulness) lives on.

If you're an old bit-head like me, you know paper tape was a viable means of transferring data to and from your computer! To add insult to injury here, some time ago, I remember seeing a paper tape reader/punch designed to be used with the early personal computers.

OK, I hear ya, "Fred, what's with this paper-tape tale? Get to the point."

TALE OF THE TAPE

For those of you too young to remember, Photo 1 shows you some actual paper tape. Note the holes. There are two types and sizes of holes found on normal paper tape-data holes and sprocket holes.

The idea is to move the paper tape through a reader or punch and interpret or create the data-hole pattern. The tape to be acted on resides on a source spool and is reeled onto a take-up spool much like modern magnetic tape.

Motors that do not contribute any timing or synchronization drive the spools. Since the data is usually destined for some kind of digital manipulation, some type of sync and timing must be provided to obtain predictableand reliable data transfer. That's where the sprocket holes come in.

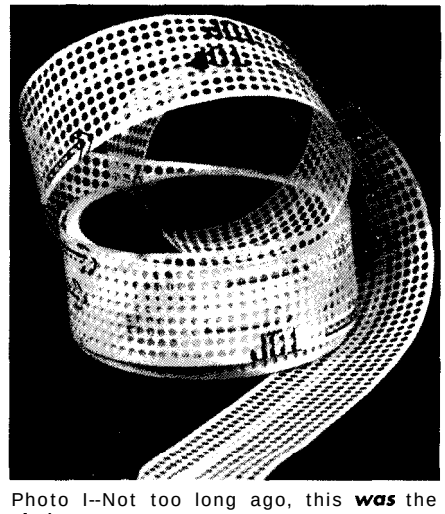


Photo I--Not too long ago, this ***was*** the ***"holey"*** grail.

Remember I mentioned that the data and sprocket holes were different sizes? The data holes are a bit larger than the sprocket holes. And, there's a good reason for that.

Looking closely at Photo 1, *you* can see that the smaller sprocket holes are centered on each line of data holes. This orientation of tape holes provides the mechanical version of data set-up and hold times with respect to a clock pulse. In the case of paper tape, the clock pulse is the sensing of the sprocket hole, and the set-up and hold times are provided by sensing the position of the larger data holes.

From a holey point of view, here's what really happens. As the paper tape is pulled across the optical or mechanical hole sensor, the data holes are sensed just a bit sooner than their corresponding sprocket hole. This bit of time between the sensing of the data hole versus the sprocket hole is the set-up time.

As the tape progresses, the sprocket hole falls under the sensor, and the data holes, which are already over the sensing area, are read. Of course, the tape is still moving, and the sprocket hole eventually exits the sensor area. Although the sprocket hole is beyond sensing range, the larger data holes are still being sensed.

You guessed it-this is the mechanical version of data hold time. It's just like the stuff you read on electronic component datasheets every day. The only difference is that, in thecaseof papertape, it's mechanically induced. Figure 1 gives you a digital view of sprocket- and data-hole timing.

Sprocket holes also serve another important purpose-error detection. As you'd imagine, the mortal enemy of any paper tape is a rip right down the old sprocket hole. Ouch!

If damage occurs to the sprocket hole area of paper tape, the set-up and hold timings I just talked about become bogus. In the case of a ripped sprocket hole, the receiving machine sees data with never-ending sprocket holes.

As a result, the data isn't synchronized, and anerrorcondition isgenerated. Ifthey're severe enough, rips in the data-hole area cause similar timing irregularities.

In addition to the monitoring of data-hole timing, parity schemes are also employed to ensure data integrity.

Dinosaurs are long dead, but we still can find their bones. As for paper tape, Photo 2 is flesh and bone of a living dinosaur.

Unless we undergo a nuclear attack, you'll probably never be called on to do anything physical with paper tape. But, if you hang around machine shops (like I do) and those machine-head guys find out that you know how to program, you may be asked to emulate a paper-tape reader or punch for one of their babies.

You see, most of the older tooling machines used paper tape to store and execute machinecode, and therearestill a bunch of these older machines being used out there.

In fact, because of this huge market, many companies sell paper-tape emulators that retrofit to the older machines. It's true that Photo 2 captured a living Jurassic Park paper-tape reader/punch, but it's also a fact that mechanical devices die of old age.

Now that I've dragged you through Paper Tape 101, and I've told you that you can buy a paper-tape emulator, you're wondering why you're reading this.

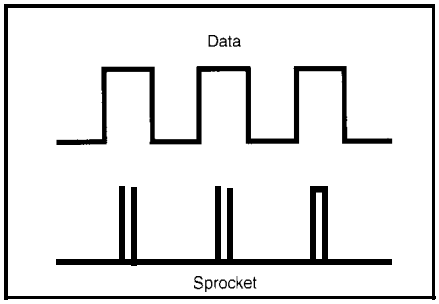


Figure I-Note the gop between the ***leading*** ond trailing edges of the sprocket pulses os they relate to the data pulses.

OK. What if you were really asked to create a paper-tape emulator? How would you go about it?

Would you machine and solder a black box full of electronics and bolt it to its milling-machine host? How would you design the interface? Would you include it physically on the black box or enable it remotely,, maybe from the machine-shop office?

What if this were your lob?! You know there's plenty of competition out there. How do you make your commodity paper-tape emulator cheap and unique?

THE HOLEY GRAIL

In Part 1, I talked about virtual front panels and how neat it would be to implement one with an embedded PC. Well, thanks to the folks at emWare, I can show you how it's done.

I discussed how emWare worked last month, so I won't go into all that here. Instead, I'll concentrate on the new features of emWare's V. 1 .1

Like V.I .0, V. 1.1 is still restricted to a single RS-232 point-to-point connection. That's OK for this application. We only need to communicate with one machine controller at a time anyway.

Also, V. 1.1 doesn't talk to Bill's Internet Explorer. Not a problem. Netscape isn't hard to obtain, and for our purposes, HTML is HTML no matter how you look at it.

There's no RS-485 support in V. 1.1, but most of the older machines don't use RS-485 anyway. So again, although a showstopper in other apps, it's of no consequence to us.

On the plus side, included in this SDK is a C-language version of emMicro. V. 1 .0 was totally 805 1 oriented. This new feature enables us to target a microcontroller other than the 805 1 The C version also includes some updates to the EMIT protocol that offer more robust function calls and events.

Also tucked in is a simulator that uses the C version of emMicro to simulate a device running on another computer. When two serial ports on the host computer are connected via a null modem cable, the simulator runs on one port and the EMITIO on the other.

You can also usetwodifferent machines if you don't have **two** open serial ports. Then via Netscape, **we** can view the simulator as a device connected to the host computer.

Guess what. We're going to plant emWare's new version on Advantech's PCM-4862 and tie it to a host PC. The PCM-4862 becomes the target controller, which inourcasewill bethepaper-tapeemulator.

Let's start by defining the problem and designing the resultant virtual front panel.

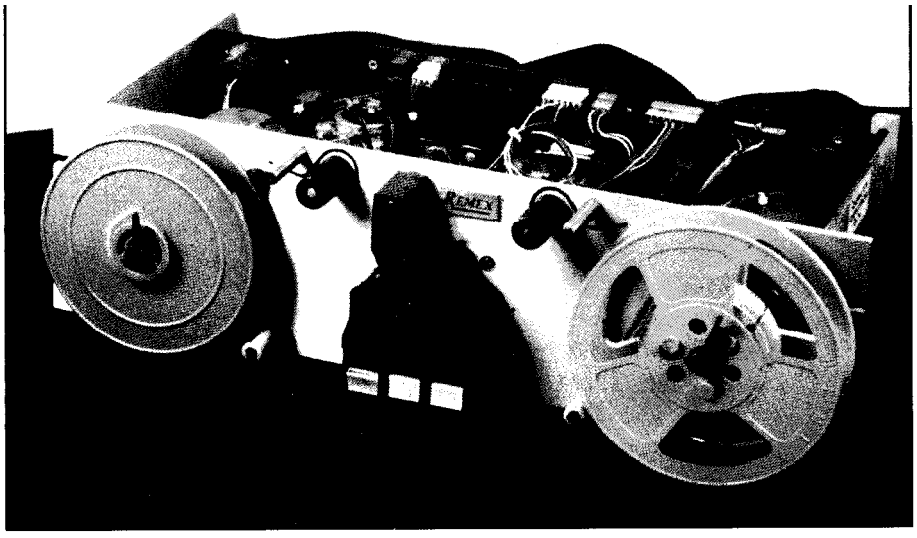


Photo 2-Mark's main thing these days is Orbiter ***parts,*** but you'd be amazed at the things ***I*** find tucked away in his shop.

The paper-tape emulator must be able to load the paper program and feed it to the milling machine. This process entails loading a G-code program from diskette to the milling-machine serial port. For this operation, I need a toad button on the front panel.

Just in case you're wondering, yep, a real paper-tape reader takes the data in as parallel and spits it out as serial. As milling machinesgo, serial is the way. Parallel data transfer would be cumbersome (and relatively expensive) in a workshop.

I'll also need to start and stop the program as necessary. So, a Start/Stop button would be good. And, it would be a good idea to be able to know what program is loaded and running on the machine. I'll add a text box for that function.

OK. We have start/stop functionality. We can identify the code we load from the PCM-4862's diskette drive. Reference the HTML in Listing 1 as I bring thecontrols to life.

First, we must code a high-level user interface for our emulator. This is done by creating an HTML page that activates the Netscape plug-in, runs the emWare Java applet emApContainer and defines its properties, defines the interface's device controls, and sets up communications between the device interface and the emulator's variables.

The first thing the HTML page does is to activate the emWare Netscape plug-in EMITJRI. EMITJRI extends Netscape with functionality specifically tailored to the needs of embedded devices.

Next,theJavaappletemApCantai ner is run. emApContainer creates the emObjects (the buttons and text area), sets their properties, and establishes the necessary communications links. emObjects are gleaned from a Java component library that ships with emWare.

Applet parameter tags determine how the emObjects behave when the application is activated. Photo 3 is the result of the HTML shown in Listing 1 -the virtual front panel for our paper-tape emulator.

Using the emWare pack utility, a file I call tapegen. h wascreatedthatcontains all of the variable, function, and event attributes, as well as static document files. To create tapegen. h, I created a separate file that describes all the variables, events, or functions for the paper-tape emulator. The emulator configuration file tapegen. i ni contains all of the definitions for the paper-tape emulator.

There are three sections that can be defined-VARS, EVENTS,and FUNCTIONS. Each section is marked by brackets ([]) around the section name.

Following the section name, I defined names and attributes of items in that sec-

listing ***1—This*** is where the real look and feel are defined.

```
<title>Circuit Cellar emWare Paper Tape Applet</title>

<hr>
<H2>Circuit Cellar Paper Tape Virtual Front Panel</H2> <BR>

<!--This EMBED TYPE tag is required to activate emWare Netscape Plugin -->

<EMBED TYPE=application/x-emj name="emitjri" HIDDEN>

<!--emApContainer is the Container Applet for emObjects. Width and
      height must be specified with the code declaration.'-->

<applet code=emApContainer.class name="tape" width=200 height=200>

<param name="DEVICE ID" value="1">
<param name="MANUFACT ID" value="1">

<param name="LAYOUT" value="none">
<param name="OBJECTS" value="4">
<param name="BACKGROUND COLOR" value-"gray">

<PARAM NAME="OBJECT0" VALUE="emJava.emSwitches.emSwitch">
<param name="OBJECT0 Reshape" value="110 10 70 30">
<param name="OBJECT0 BackgroundColor" value="darkgray">
<param name="OBJECT0 ForegroundColor" value="white">
<param name="OBJECT0 ActiveInactiveLabel" value="Run Tape">
<PARAM NAME="OBJECT0 JriVariable_0" VALUE="Run ActionEvent">

<PARAM NAME="OBJECT1" VALUE="emJava.emSwitches.emSwitch">
<param name="OBJECT1_Reshape" value="110 80 70 30">
<PARAM NAME="OBJECT1 BackgroundColor" VALUE="darkgray">
<PARAM NAME="OBJECT1 ForegroundColor" VALUE="white">
<param name="OBJECT1 ActiveInactiveLabel" value="Stop Tape">
<PARAM NAME="OBJECT1 JriVariable 1" VALUE="Stop ActionEvent">

<param name="OBJECT2" value="emJava.emDisplays.emLEDDisplay">
<param name="OBJECT2 Reshape" value="20 10 70 100">
<param name="OBJECT2 BackgroundColor" value="darkGray">
<param name="OBJECT2 ForegroundColor" value="white">
<param name="OBJECT2 NumberLEDs" value="2">
<param name="OBJECT2 LEDColor 0" value="green">
<param name="OBJECT2 LEDOn 0" value="true">
<param name="OBJECT2 LEDOn 1" value="true">
<param name="OBJECT2 LEDLabel 0" value="RUNNING">
<param name="OBJECT2 LEDLabel 1" value="STOPPED">
<param name="OBJECT2 Insets" value="10 2 5 2">
<PARAM NAME="OBJECT2_JriVariable 0" VALUE="Running ActionEvent">
<PARAM NAME="OBJECT2 JriVariable 1" VALUE="Stopped ActionEvent">

<PARAM NAME="OBJECT3" VALUE="emJava.emAWT.emTextField">
<param name="OBJECT3 Reshape" value="20 150 170 20">
<PARAM NAME="OBJECT3 Columns" VALUE="10">
<PARAM NAME="OBJECT3 JriVariable 0" VALUE="Text">

</applet>
<hr>
```

listing 2-This text is ***really*** just setting bits. For instance, ***VARARRAY sets*** bit 4, ***while VARNONE*** doesn't set any.

```
[VARS]
Run=VARNONE,VARBYTE|VARREADABLE|VARWRITEABLE
Stop=VARNONE,VARBYTE|VARREADABLE|VARWRITEABLE
Running=VARNONE,VARBYTE|VARREADABLE|VARWRITEABLE
Stopped=VARNONE,VARBYTE|VARREADABLE|VARWRITEABLE
Text=VARNONE,VARARRAY|VARREADABLE|VARWRITEABLE
```

tion. The first section shown in Listing 2 is the VARS section that defines all the variables for the emulator.

Each variable has a name and attributes identifying it to EMITIO. The attribute names are defined in the file bi tdefs. h. Each attribute can be OR-ed together (|) to generate combinations of attributes.

I already created the HTML interface for theemulatoranddefined theattributeswithin tapegen . i n i Next, I package this information together to create a static data table using the packing utility package- 1-1.

package- 1-1 reads and compresses all the files in the HTML directory. It parses tapegen. ini tocreatetheattributetables. The output is then formatted in C and written to the file statictable.h.statictable.h must be included with the rest of the project files for emMicro to link.

Thevariablesand functions predeclared in emmicroc . h must be supplied to emMicro for it to work properly. These functions enable emMicro to communicate with the embedded PC's serial port. The simulator defined these functions in simcomm . c.

To make it all come together, the C code, either from within the main calling loop or from within an interrupt, must call the emMicroentry() function to enable communications. emMicroentry()is designed to not block the current process and not consume extended processor time to execute. Now, it's time tocompile and link.

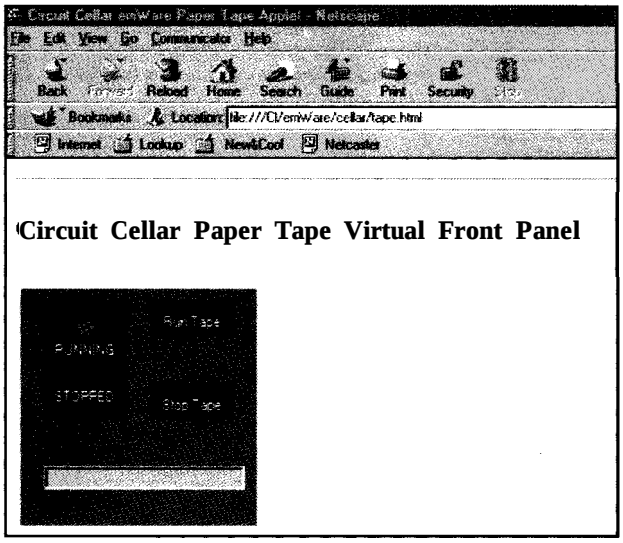


Photo 3—I added running lights and a caption just to spruce things up.

PACKING TAPE

Everything compiled and linked OK. (Of course!) Let's turn on the simulator.

Bill's Windows 95 is on the Advantech along with the tape. exe file I just compiled. Using a null modem cable, I connect the PCM-4862 to one of the beasts here in the shop with Netscape on it. Because we're using the simulator, emWare must be installed on both the embedded Advantech and the desktop PC.

After firing up EMITIO on the desktop, I issue the command to start tape. exe on the Advantech embedded PC. Finally, I start Netscape on the desktop and enter em : i nf o in the URL area

If all is well and everyone's talking, Photo 4 appears. This screen contains the data pertinent to my program variables and points to the virtual front panel already created. After clicking on the hyperlink, just point and click to use the paper-tape emulator.

TAPING IT SHUT

Although this application was pretty straightforward, imagine the possibilities. With this setup, I could remotely load and control any milling machine on a shop floor.

This emWare version doesn't support modem traffic, but when that piece is included, remote will not be just from the shop office. Remotecould bevirtuallyanywhere. By modifying the C syntax to match other processors, the C version of emWare can be put on most any embedded platform.

After all the limitations of emWare are overcome, theembeddedprogrammerand system designer will be privy to a whole new world that many thought would never come to the embedded level. And, by the way, the last word on emWare: it's not complicated, and it's embedded! \|PC|PC

Photo 4-The hyperlink doesn't usually *show up here, but with a little HTML* magic, I *saved the* machinist a *few URL keystrokes.*

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.* corn.

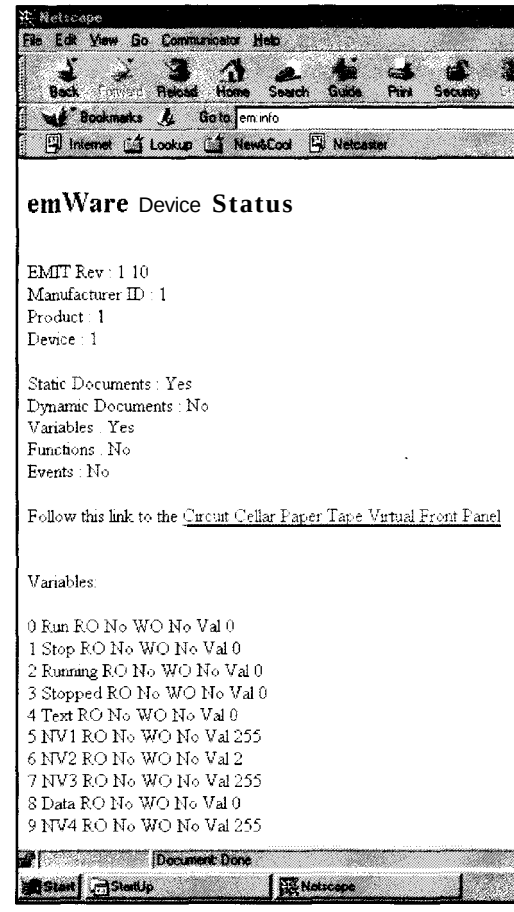
SOURCES

EMIT, SDK
emWare
1225 E. Fort Union Blvd., Ste. **220**
Midvale, **UT 84047**
(801) 256-3883
Fox: (801) 256.9267
www.emware.com

PCM-4862 All-in-One SBC
American Advantech Corp.
750 E. Arques **Ave.**
Sunnyvale, CA 94086
(408) 245-6678
Fox: (408) 245.8268
www.advantech-usa.com

IRS

416 Very Useful
417 Moderately Useful
418 Not Useful



FEATURE ARTICLE

Cheng-Yang Tan

A Hardware Keyboard Remapper

The search for the perfect keyboard—that's Cheng-Yang's mission. His PC/AT keyboard gives him the tactile feel he wants, but the keys are in the wrong place! However, with this remapper, he gains the best of both worlds.

Another flame sits ready to be sent off into the ethereal world of Usenet. I hit the Enter key hard.

Crack! It breaks off from the keyboard. Ah well, it was about time for another keyboard anyhow.

As I wait for the superglue to dry, I look through various mail-order catalogs. No luck. Computers have become such commodity items that good keyboards are a rarity. I haven't seen one that satisfies the criteria of remappability and feel.

Like operating systems and editors, the feel and position of the keys border on religion for many people. As for my broken keyboard, its feel can be best described as typing on marshmallows-not too pleasant.

Its saving grace is its remapping capabilities. I have CTRL remapped to Caps Lock, ESC remapped to tilde, plus a few others set to confuse any Nosy Parker trying to use my computer.

The best keyboard I ever used is the original IBM PC/AT 101-keyboard. Its bulwark spring technology imparts a tactile feedback no other keyboard comes close to reproducing. But, its keys are in the wrong place.

There is, of course, a software solution to this problem. My OS of choice is OS/2, and a keyboard remapper is available. But, it only works in Presentation Manager, OS/2, and DOS full-screen and windowed sessions. It does not work in WinOS/2. Also, my OS/2 box acts as an X-Server, so I have to set up the appropriate xmodmap files.

I don't like having a program sitting in my OS/2 box stealing memory and CPU clock cycles. Elegance demands a hardware solution that sits between the keyboard and PC.

Easy to implement, right? Wrong! Murphy always has his pesky fingers on the key of things, and this time is no exception.

TIMING IS EVERYTHING

The history of the PC keyboard is littered with hacks, as Ed Nisley so nicely covered (*INK* 59-61). Suffice it to say, timing and handshaking between the PC and keyboard is crucial to the success of my keyboard remapper.

Let me first go into the details of the timing, handshaking, and scan code protocols. Problem is, each and every FAQ I've read contradicts the others.

Even Ed's columns didn't seem to agree with John Dybowksi's (*INK* 57–58). John omitted any discussion of the keyboard extended codes. He didn't need them in his application, but I do.

So, what's the story? Armed with a logic analyzer, I decided to go to the source-the keyboard and the PC. In this article, I'm only interested in the PC/AT keyboard protocol. (The XT keyboard protocol is another can of worms and is well-covered in Ed's articles.)

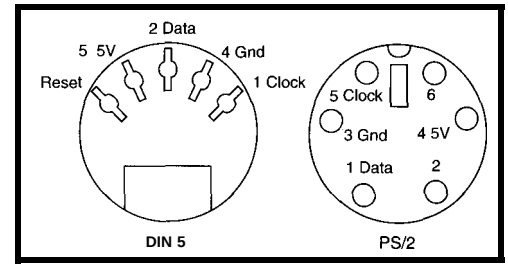


Figure 1— *Shown here are two types of keyboard connectors-D/N 5, which appears on the '486/33 and older PCs, and fhe PS/2-style connector, which seems to be the standard these days.*

THE CONNECTION

Five lines go between the keyboard and PC. Older PCs use a DIN5 connector, while newer ones use a PS/2-type connector as in Figure 1.

Most important are the CLK and DATA lines. Both are bidirectional open-collector outputs, and they're both high when the keyboard and PC are idle. With the logic analyzer hooked to these two lines, I'm ready to delve into the world of PC-keyboard communications.

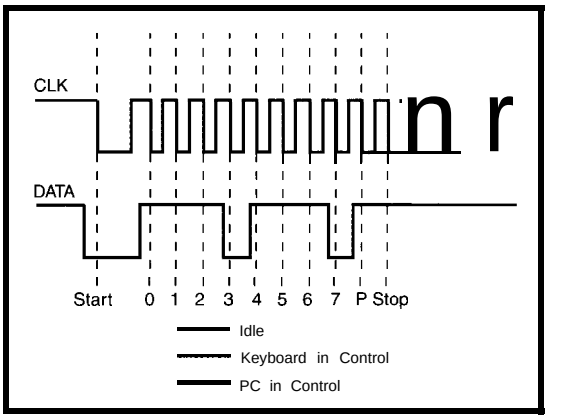


Figure 2—*DATA is clocked into the PC during each high-to-low transition. In this example, 0x77 with parity set to 1 is sent from the keyboard to the PC.*

KEYBOARD CALLING PC

First, I'll cover the simpler of the two scenarios-the keyboard talking to the PC. For this to begin, both the CLK and DATA lines must be high (see Figure 2).

The keyboard pulls the DATA line low to signal a start bit and then pulls CLK low so the PC can clock in this bit. The PC only clocks in data when the CLK line goes from high to low. DATA must be stable before and during this transition.

Next, the eight data bits plus one parity bit are clocked out. The lowest-order bit in the data is sent first, and the parity bit is set according to the number of zeros in the data bits. If the number of zeros is even or zero, the parity bit is set to one. Otherwise, it's set to zero.

Finally, the keyboard sets a stop bit that's always high. Once the PC receives the stop bit, it pulls CLK low to tell the keyboard to stop sending data. The PC releases CLK when it finishes processing the data.

PC CALLING KEYBOARD

More interesting, however, is when the PC talks to the keyboard. According to the specifications, this situation can happen anytime, even during data transmission from the keyboard to the PC, as illustrated in Figure 3.

To signal the keyboard that the PC wants to send data, it holds CLK low for ~400 µs and must then pull DATA low. This action can be thought of as the start bit coming from the PC.

The PC releases the CLK line while holding the DATA line low. The keyboard then pulls the CLK line low to signal the PC to start sending data bits. The CLK line is now controlled by the keyboard, which sets the rate of high-to-low clock transitions.

The PC sends out eight data bits and one odd-parity bit, which is clocked into the keyboard whenever there is a high-to-low transition. Each data bit from the PC must be stable sometime before each high-to-low transition.

After the parity bit is received, the keyboard holds both CLK and DATA low. Once it finishes processing the data, it releases them. Notably, there is no stop bit anywhere in this exchange!

SCAN CODES

So, what are the scan codes sent between the PC and keyboard? It turns out that the keyboard can support at least four different sets of scan codes.

The simplest, which sanely maps each unique key on the keyboard to one unique scan code, is not used for PC-keyboard communication, as Ed discussed. Instead, we have to understand the more complicated set. Table 1 lists the make scan code for each key.

You'll notice that key numbers 60 and 62 are both named Alt on the keyboard but return different scan codes. The left Alt key returns 0x11, while the right Alt key returns 0xE0 and then 0x11. The value 0xE0 signals the PC that the scan code is extended and another scan code is on the way.

Extended codes are primarily used to differentiate between two keys that usually have the same effect. Scan-code nightmares arise for the PC when it has to interpret Print Screen (key number 124) and Pause (key number 126).

The break scan code is the make scan code preceded by 0xF0 for nonextended scan codes. For example, the make scan code of A is 0x1C, so the break scan code becomes 0xF01C.

For extended codes, 0xF0 is embedded in the make scan code. For example, the Enter key on the numeric keypad (key number 108) has a make scan code of 0xE05A and a break scan code of 0xE0F05A.

There are more complications I won't deal with here since they're irrelevant for the keyboard remapper. If you're interested, see Ed's splendid series on the subject.

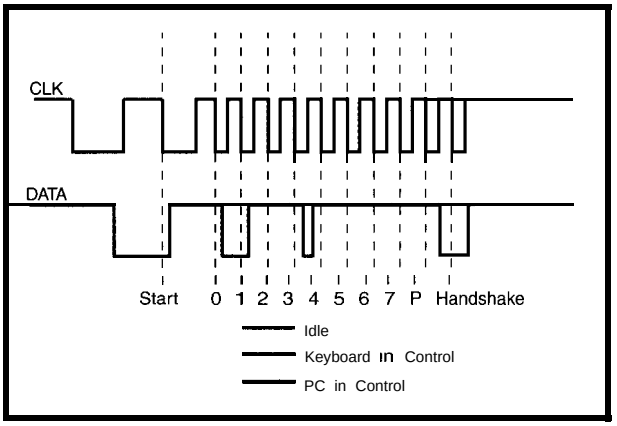


Figure 3—*The PC tells the keyboard that it needs to send data by pulling the CLK line low first. The clock is generated by the keyboard, and data is clocked in every high-to-low transition. When the keyboard has clocked in the 8 bits plus 1 parity bit, it pulls DATA low to handshake. Here, 0xFA is sent to the keyboard.* Notice *there's no stop bit!*

REMAPPER DEMON

Once I understood the PC-keyboard timing protocol and the scan codes that are passed, I could figure out how the keyboard remapper should work. I decided on a most trivial solution. The keyboard remapper hardware sits between the keyboard and the PC and

Key #	Scan Code	Key #	Scan Code	Key #	Scan Code	Key #	Scan Code
1	OE	2%	5B	54	49	99	70
2	16	'29	5D	55	4A	100	7c
3	1E	30	5%	57	59	101	7D
4	26	31	1C	58	14	102	74
5	25	32	1B	60	11	103	7A
6	2E	33	23	61	29	104	71
7	36	34	2B	62	E011	105	7B
8	3D	35	34	64	E014	106	79
9	3E	36	33	75	E070	108	E05A
10	46	37	3B	76	E071	110	76
11	45	3%	42	79	E06B	112	05
12	4E	39	4B	80	E06C	113	06
13	55	40	4c	81	E069	114	04
15	66	41	52	83	E075	115	oc
16	OD	* *42	5D	84	E072	116	03
17	15	43	5A	85	E07D	117	OB
18	1D	44	12	86	E07A	11%	83
19	24	* *45	61	89	E074	119	OA
20	2D	46	1A	90	77	120	01
21	2c	47	22	91	6C	121	09
22	35	48	21	92	6B	122	78
23	3c	49	2A	93	69	123	07
24	43	50	32	95	E04A	124	E012E07C
25	44	51	31	96	75	125	7E
26	4D	52	3A	97	73	126	1477F014F077
27	54	53	41	98	72		

* only keyboards with 101 keys USA (and others) ** only keyboards with 102 keys UK (and others)

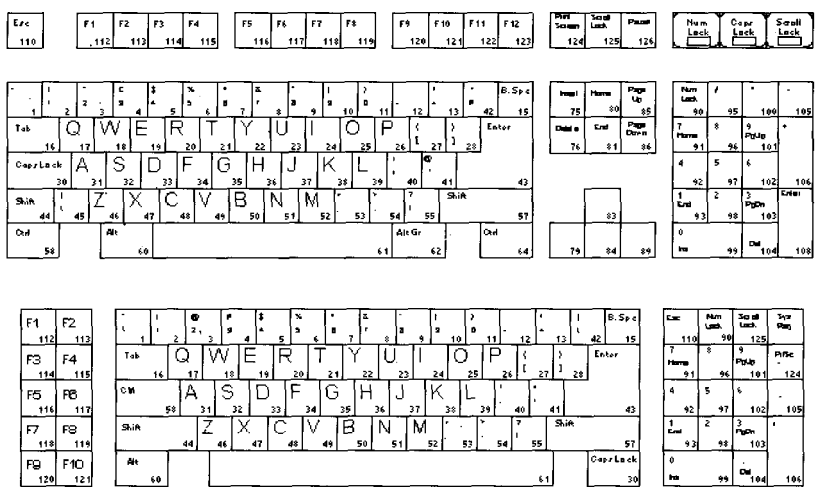


Table I--This *table* **(with** *102-* **and** *84-key keyboards) shows the mapping between the keys and scan codes. I don't have any* **information** *about the extra keys that come with the Win95-type* **keyboards.**

acts like a little demon looking for scan codes coming from the keyboard.

My remapper demon uses the scan code as the address for the entry into a look-up table. The remapper demon then takes whatever value is contained in that address and spits it back to the PC.

If the message is from the PC, the remapper demon just passes the message unscathed from the PC to the keyboard. Voilà, a remapper demon that should work for any PC OS, any PC, and any PC/AT keyboard!

With this germ of an idea, I needed to know how fast the remapper demon must move. When I press and release a key, the time between the end of a typical make scan code to the beginning of a break scan code is -2 ms.

But, the length of time needed to send a scan code is 1 ms. In other words, I have less than 1 ms to remap the scan code before spitting it back out.

Let's be conservative and assume a maximum look-up time of 0.5 ms. If I need -20 machine instructions to look up the scan code and do some housekeeping, the instruction cycle should be 25 µs or 40 kHz.

If the processor uses four clock cycles per instruction cycle, then its clock speed only needs to be 160 kHz! This simple back-of-the-envelope calculation shows that a chug-a-long microcontroller running faster than 1 MHz is more than sufficient to handle demon duties.

With the possible candidates for demon duties now wide open to all the microcontrollers ever made, I need to choose a winner. I want something small, which rules out the 8031 family and any other 40-pin behemoths.

I want something that doesn't need an external EPROM, which eliminates many micros. I want quick turnaround to compensate for my poor programming capabilities, thereby excluding those with onboard EPROMs.

Thus, one of the few candidates that survives my stringent criteria is the 18-pin PIC16C84 with its onboard EEPROM. With that question settled, I drew up the demon's schematic in Figure 4.

THE HARD DEMON

With the remapper demon built around the 16C84, I created my prototype to run at 10 MHz [it's overkill, but I like speed).

PORT A connects to diagnostic LEDs, which can be omitted if you wish. Whenever there is successful communication between the keyboard and PC, the KEYCOMM_ok or the KEYTRANS_ok LED is toggled appropriately to show that the remapper demon did its job. When the remapper demon fails in its duties, either or both of the STARTBIT_error or PARITY_error LEDs is toggled.

By "toggled," I mean that the LEDs light up if previously unlit and unlight if previously lit. If you don't like this action, modify the source code.

PORT B can connect to either the PC or keyboard directly or via PNP transistors wired as open-collector outputs. I'll use the PCD_* lines as examples because the KBD_* lines behave in the same way.

The PCD_out line is configured as an output line connected to the base of Q1. Data from the PIC to the PC is sent out via this line. I chose this method because the DATA line of the PC is open collector and normally high.

The PCD_mon line is at high impedance and connected directly to the DATA line of the PC. It monitors the DATA line on the PC side. The PCCLK_out line sends clock pulses to the PC,

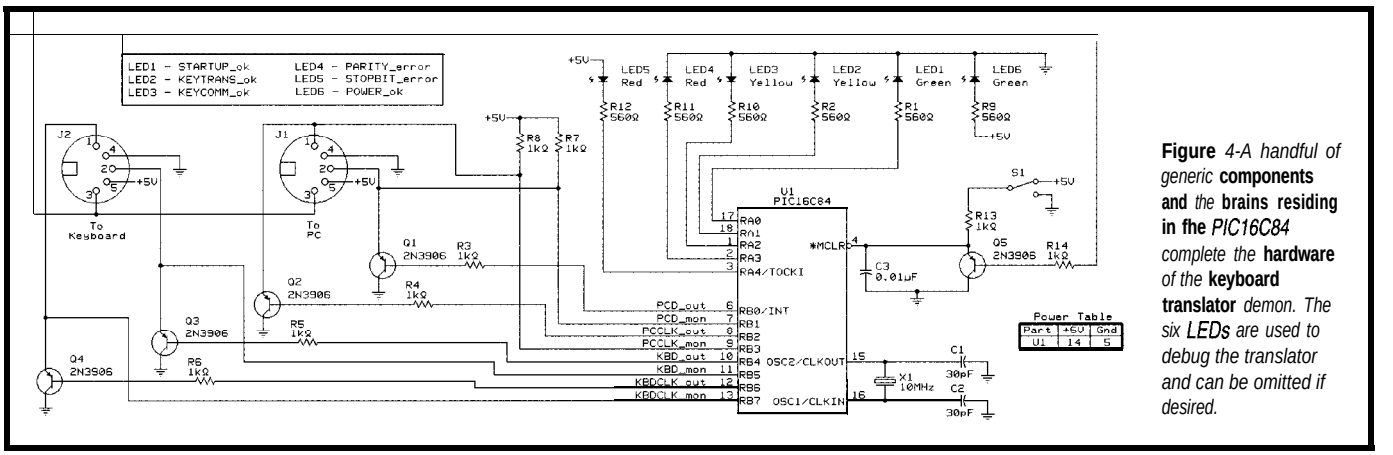


Figure *4-A handful of generic* **components and** *the* **brains residing in fhe** *PIC16C84 complete the* **hardware** *of the* **keyboard translator** *demon. The six LEDs are used to debug the translator and can be omitted if desired.*

and the PCCLK_mon line monitors the CLK line on the PC side. Similar lines are connected from the PIC to the keyboard side and have names prefixed with KBD.

The capacitor C3 prevents random resets of the '16C84, which plagued my first prototype. In retrospect, I should have known the PC supply is extremely noisy. I found this out by putting a scope on it, which of course, any seasoned PC-peripheral builder would have told me. Live and learn!

THE SOFT DEMON

With the hardware safely out of the way, I had to get the smarts into the PIC. As Figure 4 shows, I opted not to use hardware interrupts, since I can poll the PCCLK_mon and KBDCLK_mon lines to death. Figure 5 gives the flowchart of the algorithm.

There are two possible cases to think about before the demon springs into action. The first is when the PCCLK_mon line goes low. The demon goes into pass-through mode if the PCD_mon line goes low some time after PCCLK_mon line goes low. If the PCD_mon line doesn't go low, it's a false alarm and the demon goes back into polling mode.

If there's a real message from the PC, the demon simply echoes the keyboard clock obtained from the KBDCLK_mon line to the PCCLK_out line and the data from the PC from the PCCL_mon line to the KBDD_out line. Handshaking between the PC and keyboard is done as described earlier.

The other case is when the KBDCLK_mon goes low. The demon goes into translation mode and clocks in the data.

For safety, it checks that the data clocked in has the correct parity and also matches the parity bit sent by the keyboard. If this fails, it toggles the PARITY-error LED. It also checks that it gets a stop bit. If this fails, it toggles the STOPBIT_error LED.

Once it thinks it has the data (whether it's correct or not), it pulls the KBDCLK_out line low to stop the keyboard from sending more data. If the data is 0xF0, it simply echoes it to the PC. If the data is 0xE0, it marks the data as extended, echoes 0xE0 back to the PC, and waits for the second half of the extended code.

After receiving this, the data is echoed without change back to the PC. If the data is just an ordinary scan code, it looks up the value in the look-up table using the scan code as the address.

When the lookup is done, the value from the look-up table is echoed to the PC. Once the exchange with the PC is over, the demon releases the KBDCLK_out line and returns to polling mode.

In translation mode, the demon always keeps an eye on the PCCLK_ mon line in case the PC wants to talk. If it does, the demon immediately abandons translation mode and enters pass-through mode.

In the many hours that I've had my prototype connected to my PC, I've never experienced this scenario. However, you can never be sure what Murphy is up to!

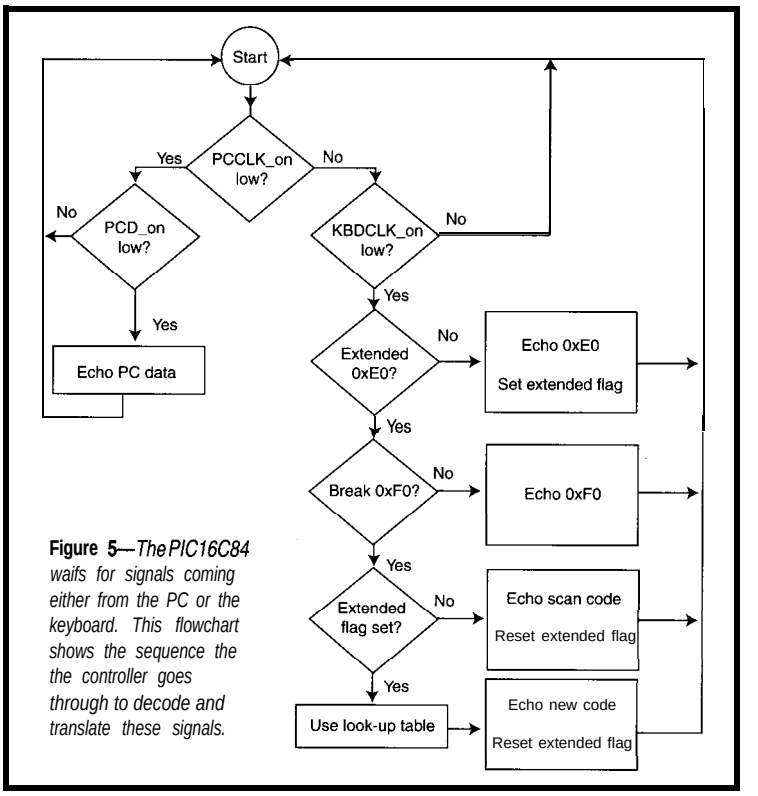


Figure 5—*The PIC16C84 waits for signals coming either from the PC or the keyboard. This flowchart shows the sequence the the controller goes through to decode and translate these signals.*

TESTING

I tested the keyboard remapper on a Gateway 2000 with a Micronics motherboard, a Micron PC P5, and a no-name clone with an AMI motherboard.

They used three different keyboards-the original IBM PC/AT 101-key keyboard, the Micron keyboard, and a generic no-name keyboard. All worked flawlessly with the keyboard remapper.

ENHANCEMENTS

Naturally, some enhancements can be made to my remapper demon.

As you can tell, it doesn't currently remap extended scan codes. I believe this problem can only be solved with more complicated hardware.

The problem arises if you want the remapper to be universal. Universality would require, in particular, the remapping of an extended key to any other key and vice versa.

For example, suppose you want to remap the right Alt key (scan code 0xE011) to the A key (scan code 0x1C). When you hit the right Alt on the keyboard, the make scan-code translation presents no problem to the demon.

However, when the break scan code is to be translated, the demon has to wait 1 ms for 0xE0, 1 ms for 0xF0, and another 1 ms for 0x11 for a total of 3 ms. It then takes another 2 ms to send out 0xF01C as translation.

But, 5 ms is just too long because another key can be pressed during this time. Because the translator isn't ready to receive a new key, this key is stored in the memory buffer on the keyboard.

Worst of all, because in general all keys can be extended (i.e., scan codes may all take between 4 and 5 ms to process), more and more scan codes need to be buffered. The small 8-byte keyboard buffer quickly gets swamped and overflows. The obvious hardware solution is to build a remapper with some RAM so a large ring buffer can be used to queue the keys to be remapped.

Another possible enhancement is to allow reprogramming of the look-up table in situ. The 16C84 can be programmed with two lines-one DATA and one CLK. And, that's exactly what I have sitting between the PC and the remapper. Some clever hacking of the 8042 keyboard controller in the PC to generate the clock and data for programming is a reasonable possibility.

NOW THAT I GOT IT...

So, how long did this simple project take from conception to realization? My original estimate was 1-2 weeks. It turned out to be 12.

The main stumbling block was bad documentation. Once that was overcome, my inexperience with the PIC became the new stumbling block. Blowing up a test PC certainly didn't help.

I had quite a bit of fun with this project. But now, with OS/2 WARP 4 installed on my PC and its voice-recognition success rate at greater than 95%, I might just throw away the keyboard and simply talk to my computer! ❑

Cheng-Yang Tan received his Ph.D. in physics from Cornell University. His interests include linear accelerators and computer simulations on supercomputers. You may reach Cheng-Yang at cytan@fnal.gov.

SOFTWARE

Microchip Technology's free compiler and simulator are available at www.microchip.com. Scan codes may be downloaded from Altek Instruments at www.hello.co.uk/altek/scan_doc.html. A keyboard remapper for OS/2 is available at hobbes.nmsu.edu as kbdredef.zip. Source code can be downloaded from the Circuit Cellar Web site.

SOURCE

PIC16C84
Microchip Technology, Inc.
2355 W. Chandler Blvd.
Chandler, AZ 85224-6199
(602) 786-7200
Fax: (602) 786-7277
www.microchip.com

419 **Very Useful**
420 **Moderately Useful**
421 **Not Useful**

DEPARTMENTS

Applying Direct Digital Synthesis

MICRO SERIES

Tom Napier

Making Waves with NCOs

Part 1 of 2

Are you tired of the limited output range of your VCO? What about its difficult modulation? Tom suggests you try an NCO, which lets you synthesize any waveform and offers you high-frequency output.

Wouldn't it be nice if anytime you needed an accurate sine- or square-wave signal, you could dial up exactly what you wanted, whether it was a few hertz or several megahertz?

Well, you can with a Numerically Controlled Oscillator (NCO).

The traditional device for generating a sine wave with an accurately known frequency is the Voltage Controlled Oscillator (VCO). Its frequency is set by dividing its output and comparing it to a crystal reference, commonly 1 kHz.

The phase difference between the divided output and the reference generates an error signal that changes the control voltage driving the oscillator.

The VCO's output frequency can be changed over a two-to-one range by changing the division ratio, but it's not useful if you want a wide range of output frequencies plus low distortion.

A 100-MHz VCO won't give you an audio frequency output, and wide-range VCOs tend to have a high second-harmonic content. They are also difficult to modulate accurately.

WHAT'S AN NCO?

A signal generator based on an NCO uses Direct Digital Synthesis (DDS). In other words, the NCO chip generates a series of numerical outputs which, when converted to analog by a DAC, become the desired sine-wave output.

Of course, you can synthesize a sine wave-or any repetitive waveform-by using a computer to output the contents of a look-up table at regular intervals. Computer sound cards do just that.

However, to change the frequency, you either have to recompute the table or change the sampling rate. Neither technique is convenient for frequencies above the audio range.

The NCO reverses the process, keeping the sampling rate fixed but changing the size of the amplitude step between samples. Figure 1 shows what's inside a typical NCO chip.

The NCO contains a wide (2448 bit) accumulator. The number in the accumulator is incremented at a fixed rate, generating an output that ramps from zero to full scale and then wraps back to zero. A look-up table, or its algorithmic equivalent, converts each ramp into a sine-shaped output as in Figure 2.

The NCO contains a register you can load with any desired increment number. The bigger the increment, the higher the output frequency.

SETTING THE FREQUENCY

If you add 1 to the accumulator every microsecond, you get a very low-frequency ramp and thus a low-frequency sine wave. For example, if the accumulator has 32 bits, you get one output cycle every 2^{32} µs (i.e., 4295 s). Generating an accurate 72-min. sine wave is otherwise pretty tough, but most of us want higher frequencies.

If you put 42,949,673 in the increment register, then 42,949,673 is added to the accumulator every microsecond. The output frequency will then be 10.00000001 kHz.

With 42,949,674 in the register, the output frequency is 10.00000024 kHz. In other words, you can generate a frequency of 10 kHz with a precision of one part in 43 million.

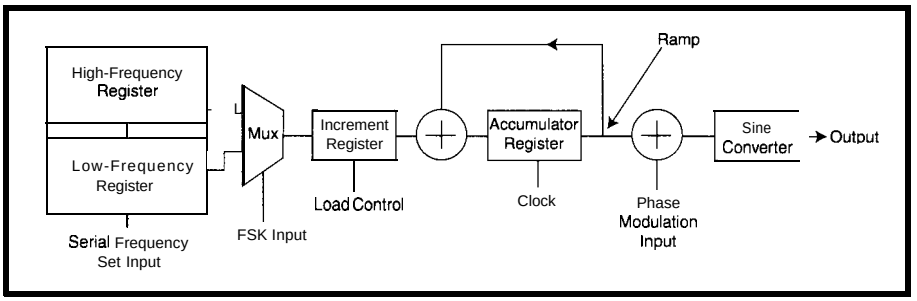


Figure I-An *NCO* **has** *a register that stores the user set frequency, an accumulator, and a ramp-lo-sine converter.*

In real life, you'd generally use a higher clock frequency than 1 MHz, since the maximum frequency that an NCO can generate is about a third of its clock frequency. Readily available NCO chips run up to -70 MHz and thus can generate outputs up to about 2.5 MHz. Special devices are available with clock rates up to 1 GHz if you really need and can afford them.

The output frequency of an NCO is:

$$\frac{\text{Clk} \times \text{Inc}}{2^L}$$

where *Clk* is the clock frequency, *Inc* is the increment you set, and *L* is the accumulator length in bits. The resolution (i.e., smallest frequency change you can make) is:

$$\frac{\text{Clk}}{2^L}$$

If your clock is 50 MHz and you use a 48-bit NCO, you can have a resolution of 0.177 µHz in, say, a 10-MHz output.

The absolute accuracy of the output frequency equals that of the clock driving the NCO. You could use a $3 crystal oscillator with an accuracy of 50 ppm, but nothing, other than cost, stops you from using a rubidium-vapor reference frequency and getting one part in 10^{12} accuracy.

The combination of high resolution and high absolute accuracy makes the NCO the ideal device for generating a tunable reference frequency.

CONTROLLING THE FREQUENCY

It's a straightforward programming job to read the user's desired frequency from thumb-wheel switches, compute the needed increment, and load it into the NCO.

Most NCOs are microprocessor compatible. They have a buffer register that's loaded one byte at a time. Once all the bytes are loaded, an update pulse is sent to the chip, transferring the buffer contents to the increment register and switching the NCO's output to the new frequency.

My favorite NCO-the Harris HSP45 102-has a serial input. Since it has only 28 pins, it's cheaper than many others. However, its maximum clock frequency is 33 or 40 MHz (depending on the grade), limiting it to about a 15-MHz output.

Programming can be made easier by picking a crystal frequency that's a power of two. My favorite frequency is 33.554 MHz, since it gives control inputs of 128 units per hertz.

Unfortunately, 33.554 MHz is a special-order frequency. 32.768 MHz is the closest off-the-shelf frequency. It gives a resolution of 131072 units per kilohertz but can't give exactly correct 1 -Hz steps.

CONVERSION TO ANALOG

An accumulator usually has 32 bits. Remember, more bits mean better frequency resolution.

You can't convert all 32 into a 32-bit sine output because no one makes a 32-bit DAC. Typical NCOs have 12-bit outputs, which is sufficient for generating a very accurate sine wave. For many

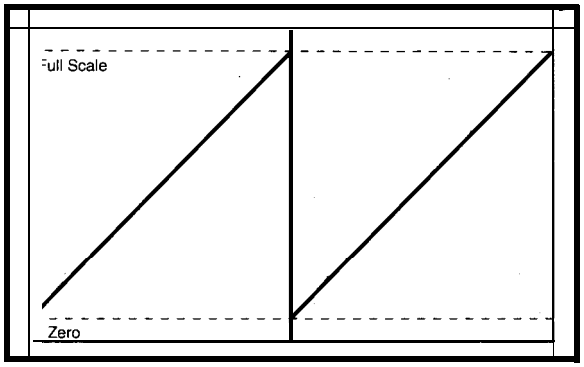


Figure 2—*The ramp samples from the accumulator are converted to the sine-wave samples, which form the output of the NCO.*

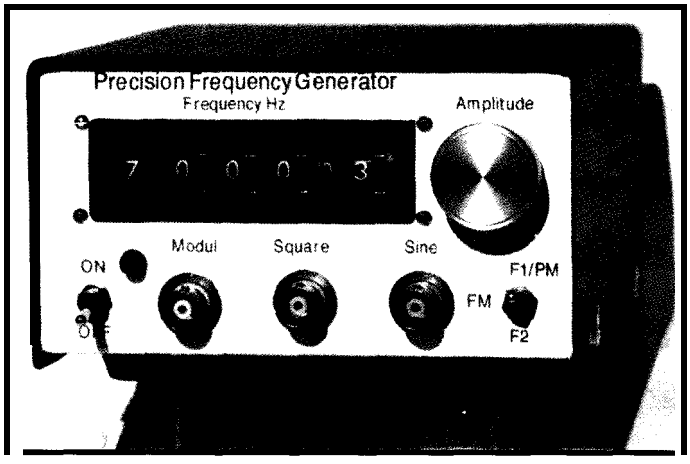


Photo 1--This *view of the completed NCO generator shows the front-panel controls and output connectors.*

applications, 8 or 10 bits are enough, particularly since an 8- or 10-bit DAC is much cheaper than a 12-bit one.

The main technical difference between a 10- and 12-bit DAC is that at low frequencies, you see 1024 little steps in the output rather than 4096. The output of the IO-bit DAC also has a higher level of spurious frequencies.

In theory, a 12-bit DAC generates spurs at a level of -74 dBc, and a 10-bit DAC at a level of -68 dBc. But in practice, noise and crosstalk in the circuit are likely to be more significant.

The DAC has to accept a new input sample every time the NCO is clocked, so it runs at some tens of megahertz.

A PRACTICAL NCO GENERATOR

The NCO generator shown in Photo 1 and diagrammed in Figure 3 uses a Harris HSP45102BIP-33 NCO chip to generate frequencies from 1 Hz to 9.999 MHz. The frequency is set by a thumb-wheel switch that has four digits plus a range switch.

Either a 32.768- or 33.554-MHz crystal can be used. The latter is preferable, since it makes all the output frequencies exactly correct.

An 18-pin PIC16C54 reads the switches, computes the increment, and loads it into the NCO. It also reads a mode switch and sets up the modulation type according to the switch position.

I used a 10-bit DAC-the Harris HI-5721BIP. It costs about $11 if you shop around. The 5731BIP, which has 12 bits, could be used, but its pinout differs.

To put costs in perspective, the NCO chip retails for about $15, and thumb-wheel switches cost around $7 per digit. Of course, if you decide you'll never need a high-frequency output, then use a slower clock and cheaper DAC. I used an inexpensive 32.768-MHz crystal and wrote the firmware for this frequency.

MODULATING THE OUTPUT

Apart from its low cost and small size, the HSP45 102 NCO is also easy to modulate.

Any NCO's frequency can be changed by loading a new increment. There will be a time lag before the frequency change occurs, but when it does, the change is instantaneous.

The new frequency starts at whatever phase the old frequency reached. There is no amplitude jump at the transition and no slewing through intermediate frequencies as there is with a VCO.

The HSP45102 stores two 32-bit increment numbers, and you can switch between them whenever you want. (To be strictly accurate, the frequency synchronizes with the crystal clock, but at low data rates, this fact is unimportant.) Thus, this NCO is ideal for implementing Frequency Shift Keying (FSK) of a carrier.

Some NCOs let you modulate the phase of the output signal. The HSP-45102 has two phase-control bits, which permit the phase to be switched instantaneously to any multiple of 90".

Thus, you can generate Binary Phase Shift Keying (BPSK) or Quadrature Phase Shift Keying (QPSK). Depending on its phase of modulation, the output may jump through anything from zero to the peak-to-peak amplitude.

Since signal amplitude can modulate by altering the reference current applied to the DAC, the possibilities for developing and testing modems and other communications devices are endless.

OUTPUT FILTERING

The DAC's output is a series of steps that take place at the clock frequency. When the output frequency is more than -10% of the clock frequency, the raw output looks nothing like a sine wave, but it's quite easy to clean up with a low-pass filter.

Figure 4 shows the DAC output spectrum. It has a line at the wanted frequency but also a line at the clock frequency minus the wanted frequency.

There are other lines around each multiple of the clock frequency, but if you can get rid of the first unwanted line, the rest won't be a problem. The filter should pass the highest frequency desired (10 MHz, in this case) and not

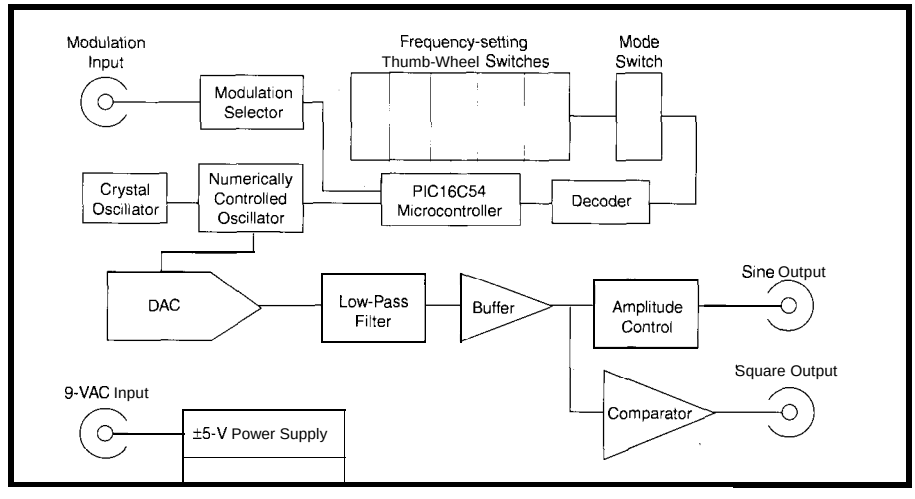


Figure 3—*The core of the generator is the NCO chip, which generates samples at the crystal frequency. These are converted into a sine wave by the DAC and low-pass filter. The output frequency is set by five thumb-wheel switches, which are read by the microcontroller.*

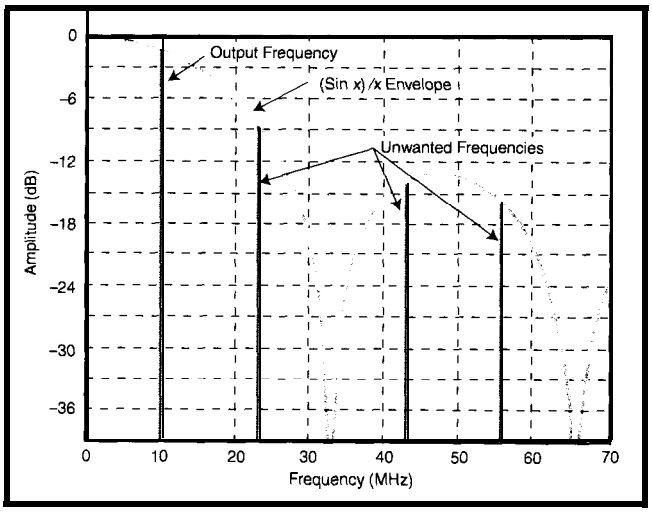


Figure 4—*The spectrum of the output* **from** *an NCO has a (sin* **x***)/***x** *envelope and contains both wanted and unwanted frequencies.*

pass too much of the lowest unwanted frequency (-23 MHz).

This filter is quite tough to design. One solution is to use a commercial which cuts off at 14 MHz. But, it is designed to work in a 50-Ω system, which is a bit low, and it cuts off so sharply that when the output phase is modulated, the filter rings.

Filter choice is a compromise between reducing the amplitude of the unwanted frequencies and producing clean modulation. A sharp cut-off filter has little effect on the wanted frequencies and attenuates the unwanted ones, but it generates ringing in the output waveform when a step takes place, as in phase modulation.

On the other hand, a filter producing a clean output transition attenuates the higher wanted frequencies but doesn't work as well with unwanted frequencies. (You can do it, but it requires a more complex filter than this application justifies.)

I used a four-pole Butterworth filter in a current input/voltage output configuration. Its cutoff is fairly sharp, but it doesn't introduce an impossible output distortion. I modified the component values from the ideal Butterworth to standard inductor values, but the difference is negligible.

This filter configuration compensates for the output resistance and capacitance of the DAC. It also works with an HI-5731 DAC, but it needs component value changes if used with a DAC that has a higher output resistance than the 227 Ω of the Harris parts.

The filter has a high output impedance, so it's followed by a buffer amplifier. Any op-amp with a unity gain bandwidth over 20 MHz can be used, provided it can drive the output load with a 1-Vp-p signal. It's convenient if it runs off 5-V power supplies. Most of the faster op-amps do.

I used a National LM6361, but many other amplifiers work as well. If you need a higher output amplitude, you can wire the amplifier to have higher gain, but you'll need a faster amplifier than the LM6361.

ADJUSTING THE OUTPUT

Since the DAC output is unipolar, a current source is applied to center the output around 0 V. (Since the output can be as low as 1 Hz, the usual solution—a coupling capacitor-didn't appeal.)

A slow op-amp monitors the mean DC output and adjusts the balance current to make the mean level equal zero. Thus, the output behaves as if it's AC coupled with a very large capacitor. The roll-off is below 0.1 Hz.

High-speed current-output DACs generally sink their output to -5 V. The output is distorted unless it always lies within the DAC's output compliance range. This limits the maximum load resistance a DAC can drive and the possible mean DC level.

For example, some video-output DACs must be loaded with 37.5 Ω, and their output compliance range doesn't go positive. Therefore, you can't just apply an offset current but must use an amplifier to shift the output to a mean DC level of zero.

The DAC reference current is adjusted with a trimmer to set the open-circuit output to 1 Vp-p. In use, the output amplitude is set by a simple 500-Ω potentiometer on the front panel. So, the output voltage isn't calibrated and it depends on the load resistance, but it's adequate for most purposes.

This solution is simpler than most alternatives. Be sure to use a cermet potentiometer, not a wire-wound one. Another tip: 300-Q TV antenna cable, if twisted, is a good way to connect the circuit board to the potentiometer.

FRONT-PANEL CONTROLS

The generator is controlled by a PIC16C54 whose clock is half the frequency of the NCO clock. Every 20 ms, the PIC reads the five thumb-wheel switches, computes the corresponding increment number, and loads it into the NCO.

Unless you change a switch position, the NCO is continuously updated with the same numbers. This has no effect on the output.

Photo 1 shows the front-panel layout. Apart from the four digit switches and one range switch, the front panel carries three BNC connectors, the usual on/off switch and warning light, and a three-way toggle switch to set the modulation mode. One BNC connector is the modulation input, one is a square-wave output, and one is the variable-amplitude sine-wave output.

The three-way switch controls the modulation input. In its up position, the frequency set by the thumb-wheel switches is loaded into the NCO as the high frequency.

When the switch is down, the indicated frequency is loaded into the NCO as the low frequency. Either way, any thumb-wheel change is reflected immediately in the output frequency.

When the switch is in its center position, the thumb-wheel switches have no effect. But, a TTL-level signal applied to the modulation connector switches the NCO between its low and high frequencies.

With no modulation input connected, it defaults to the high frequency. If both frequencies are set to the same value, the modulation has no effect.

Thus, to set up FSK between two frequencies, put the switch in its down position and set the low frequency. Then, switch it up and set the high frequency. The frequency modulation starts when you switch to the center position.

Zero frequency is a legitimate input. It produces phase coherent on/off keying.

When the switch is up, a TTL-level signal applied to the modulation connector produces biphase modulation of the generated signal. In the down position, the modulation input has no effect.

QPSK requires two modulation inputs. Since I ran out of front-panel space, it's not available in this version of the generator. I haven't hooked up amplitude modulation, either.

TIL NEXT TIME

Now that you've seen how an NCO generator works and what it can do, it's time to think about building one.

Next month, I'll cover the schematic, construction, and testing of this versatile sine- and square-wave generator. ❑

Much of what I know about NCOs and filters I learned from my former colleagues in the Signal Recovery Group of the Aydin Corp.

Tom Napier has worked as a rocket scientist, health physicist, and engineering manager. He spent the last nine years developing space-craft communications equipment but is now a consultant and writer. You may reach Tom via E-mail at *eric@voicenet.com.*

SOURCES

Harris parts
Allied Electronics
7410 Pebble Dr.
Fort Worth, TX 76118
(817) 5953500
Fax: (817) 5956406
www.allicd.avnet.com

DACs and fast amplifiers
Analog Devices, Inc.
One Technology Way
Norwood, MA 02062-9 106
(617) 329-4700
Fax: (617) 329-1241

Thumb-wheel switches, connectors, inductors, amplifiers, 74ACT chips, PIC microcontrollers
Digi-Key Corp.
701 Brooks Ave. S
Thief Falls, MN 5670 l-0677
(218) 681-6674
Fax: (218) 681-3380

HSP45102, HSP45106, HSP45116, **HI-5721BIP,** HI-5731 DAC
Harris Corp.
1025 W. Nasa Blvd.
Melbourne, FL 329 19
(407) 727-4000
Fax: (407) 724-3973
www.harris.com

PIC16C54
Microchip Technology, Inc.
2355 W. Chandler Blvd.
Chandler, AZ 85224-6199
(602) 786-7200
Fax: (602) 786-7277
www.microchip.com

PLP-10.7
Mini-Circuits
13 Neptune Ave.
Brooklyn, NY 112350003
(718) 934-4500
Fax: (718) 332-4661
www.minicircuits.com

LM6361
National Semiconductor
P.O. Box 58090
Santa Clara, CA 95052-8090
(408) 721-5000
Fax: (408) 739-9803

Harris parts
Newark Electronics
12880 Hill Crest Rd.
Dallas, TX 75230
(972) 458-2528
Fax: (972) 4582530
www.newark.com

General-purpose **NCOs**
Stanford Telecom, Inc.
480 Java Dr.
Sunnyvale, CA 94089
(408) 7452660
Fax: (408) 341-9030
www.stelhq.com

I R S

422 Very Useful
423 Moderately Useful
424 Not Useful

FROM THE BENCH

Jeff Bachiochi

Listening to Magnetism

Although some of the New Age rave would have you believe differently, you can't hear magnetic waves—until now.... Jeff builds a magnetic-field sensor that lets you know when you're getting to full strength.

Over requested trips by Cub and Boy Scout troops to a visit to Battleship Cove in Fall River, MA. My youngest son, Kristafer, and I spent last weekend aboard the USS Massachusetts.

This battleship is one of only a few permanently housed after its many years of service. Overnight groups have full rein of the ship.

Although the equipment is not in working order, it doesn't kill the kids' quest for adventure. Free time is spent exploring all the nooks and crannies.

One dad brought his cell phone so he could touch base with home Saturday night. To his surmise, he couldn't get a link anywhere in the ship.

Now, parts of the ship have in excess of a foot of steel armor plating. It's no wonder his bitty phone couldn't punch through. Once he went on deck, the connection was easily made. So much for living in a Faraday cage.

Unlike the lack of electromagnetic waves on the exterior of that tin can, our environment is chock full of emissions. We don't think of magnetic waves much because, like gravity, we can't see them directly.

Oh, we see the effects of it, but our bodies don't have sensors tuned to such properties. This month, I've created a project that makes you more aware of the magnetic forces around you.

MAGNETISM AS A FORCE

Edwin Hall first noticed magnetism's effect on the flow of current in 1879 while at Johns Hopkins University. He discovered that a magnetic field (x axis) passing perpendicularly through a current path (y axis) tended to draw or push the current perpendicular to both (z axis). This created a differential voltage (on the *z* axis) that was directly proportional to the magnetic-field density.

Hall-effect sensors are used today to measure current (via a magnetic field), to count or measure movement, and to act as an isolated switch. Because they're essentially environmentally sealed, they can be used in dirty environments without degradation.

Early Hall-effect sensors had lousy delta temperature, current, and stress properties. Constant-current drivers, temperature compensation, and improved architectural sensor layouts all contribute to today's enhanced devices. Hall-effect sensors now have linear or switched outputs that respond to either unidirectional or bidirectional fields.

Although all Hall-effect sensors begin life as linear devices, many applications require only the acknowledgment of a magnetic field's presence. So, additional Schmitt hysteresis circuitry is added to create a more user-friendly switching device.

This project requires the linear section only. If you look at Figure 1, you see temperature-regulated biasing that stabilizes this new sensor's sensitivity to temperature and voltage fluctuation.

You also see a chopper-stabilization technique for eliminating the offset voltage due to mechanical stresses. The linear sensor is available only in a SOT-89A surface-mount package, so get out your magnifier if you wish to attach leads to this device.

HAL400

No, this isn't 2002: ***A Space Odyssey***. The HAL400 is ITT's Intermetall linear CMOS Hall-effect sensor IC.

The HAL400 delivers -40 mVmT. The magnetic offset is typically held to less than 10 mV, while the noise floor is typically below 400 µV.

HAL400's differential output allows the chopper compensation to cancel,

while reinforcing the Hall-effect output. The sensor runs on as little as 4.3 V, yet can accept input up to 12 V.

An LM660 op-amp filters the differential output from the Hall sensor and converts it to a single-ended, ground-referenced output (see Figure 2). A bipolar amplifier would have done well here, but I wanted to use this single-supply amplifier for a specific reason I'll discuss shortly.

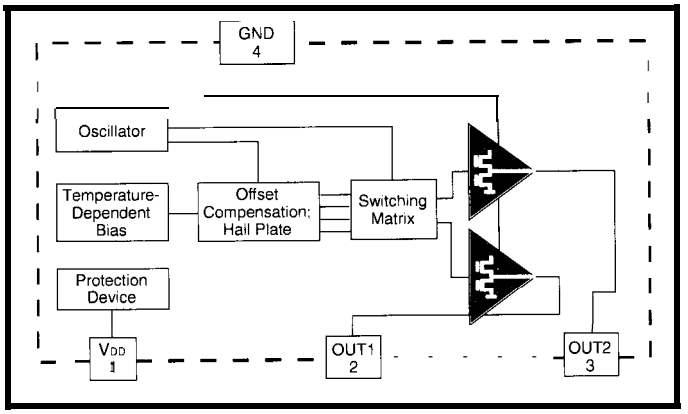


Figure 1 --*The* **HAL400 Hall-effect sensor** *has built-in temperature compensahon and* **chopper stabilization.**

Since this amplifier is single ended, the zero magnetic-field level (normally, zero in a bipolar world) is shifted to 2.5 V. This shift becomes a problem since we want to amplify the signal.

PGA204

No, this isn't a movie rating or a college course. The PGA204 is a programmable-gain instrumentation amplifier by Burr-Brown.

Burr-Brown makes a whole family of these babies. This one has lx, 10×, 100×, and 1000x gains digitally programmed through two input bits. This part is bipolar, so WC need to get rid of the 2.5-V offset added by the first op-amp.

This situation turns out to not be such a big deal. The Hall sensor has a magnetic offset, which can normally be ignored unless we want to amplify the signal with any significant gain.

By applying the first amplifier's output to the noninverting input and a 2.5-V reference to the inverting input of the PGA204, not only can the single-ended offset be subtracted, but since the reference is adjustable, the magnetic offset can also be nulled out at the same time. This adjustment is done with the PGA204 on the 1000x gain setting.

Two additional notes here. The LM336-2.5 reference diode circuit uses two silicon diodes in the potentiometer connections. These diodes significantly compensate for the output drift of the 336 due to temperature changes.

Since the PGA204 is bipolar, it requires a negative voltage to operate. A simple ICL7660 inverter produces the necessary -5 V from +5 V. This switching inverter is noisy and needs to be kept as far from the amplifier inputs as possible to reduce noise pickup.

The 204 amplifier produces output 1.5 V less than the power supplies. Using 5-V supplies means the maximum output swings are ±3.5 V.

The final amplifier-another section of the single-ended LM660—must again be offset by 2.5 V to handle the bipolar input. The LM336 output is used again to shift up the PGA204's bipolar output by 2.5 V.

The LM660's output can swing rail-to-rail (well, pretty darn close) and ensures that the output is not less than ground or greater than 5 V. This output matches the A/D input specifications for the LTC 1298.

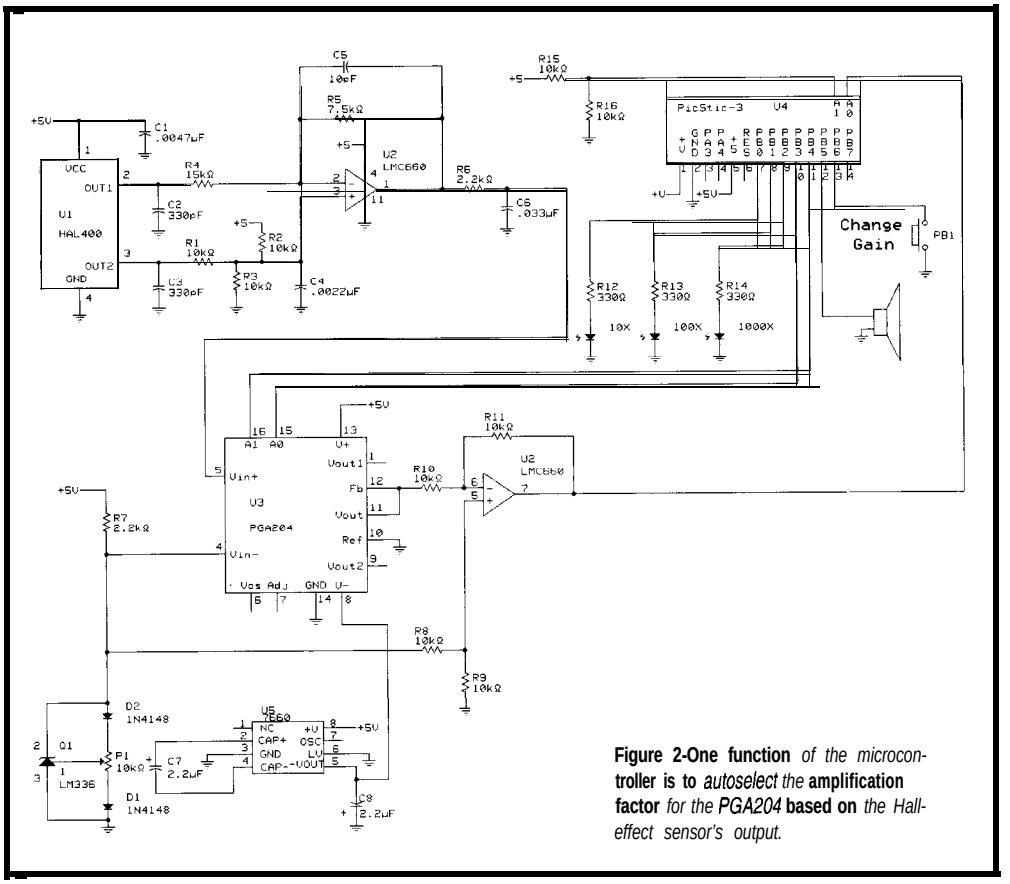


Figure 2-One function *of the microcontroller* **is to** *autoselect the* **amplification factor** *for the PGA204* **based on** *the Hall-effect sensor's output.*

LTC1298

The LTC1298 from Linear Technology is a two-channel 12-bit ADC using V_{CC} as its reference. A 2.5-V input converts to half-scale or a count of 2047 out of 4095, which is the zero magnetic level.

Any DC magnetic field pushing on the sensor causes the voltage to increase at the ADC. And, any DC magnetic field pulling on the sensor causes the voltage to decrease at the ADC. AC magnetic fields cause the voltage to rise and fall in step with the frequency of the AC field (within the bandwidth of the circuit).

Now, why add on the ADC (e.g., a PicStic) when the output

of the amplifier could run a meter or similar visual or audible device? Simply to gain more control of the output data.

Using a micro, the data can be manipulated and shipped out as ASCII serial data. The control inputs of the programmable-gain amplifier can be automatically set for the best range.

LEDs can be driven, indicating the range of the amplifier. Or, any number of LEDs can be used with a serial shift register implementing a bar graph.

I hope you find many uses for this project. All I want is some indication of magnetic-field strength. So, I'll program for audio output.

PICSTIC-3

One of the benefits of using a PicStic is being able to write fast code in BASIC, thanks to microEngineering Labs' PicBasic Compiler.

One of the commands available in BASIC is the `Sound` command. Values **1-127** produce frequency output from 95 to 1000 Hz. The duration of the tone can be fixed in increments of 12 ms.

By continuous sampling (and doing nothing else), I can get about 4000 samples per second (-250 µs per sample). A simple loop may consist of a conversion, a tone whose frequency is based on the conversion value, and a check for a button push (to change gains). Listing **1** has the BASIC code I wrote using the PicStic-3.

The code begins with the amplifier setup for a gain of 1x. None of the GAIN LED outputs are enabled. Only three are used, as none indicates a 1x gain.

Next, the ADC is sampled and the conversion value is divided by 32 (giving a number between 0 and 127.) This number is passed to the tone generator. The frequency of this tone is an indication of the magnetic field present at the sensor's sensitive area.

At the end of the tone burst, the button is sampled. If it's not pressed, the program takes another sample. If the button is pressed, the gain is cycled through 1x, 10x, 100x, and 1000x.

The GAIN LED outputs and the amplifier gain control bits are updated for each change in gain. Holding the button down repeatedly cycles through the gains. Once the button is released, sampling begins again.

The 1x gain configuration samples to a 1-mV resolution. The full conversion resolution is available to the user.

Since the tone output is broken into only 127 steps, the conversion value is divided by 32, which drops the resolution to 32 mV per tone step.

At 10x, resolution falls to 3 mV, and at 100x, it's down to 300 µV, and so on. At 1000x, the drift and the noise floor begin to rear their ugly heads.

MORE ROOM NEEDED

Originally, this project was to fit in a hand-held wand (see Photo **1**). But, discrete parts quickly filled the space. There was no room for a 9-V battery.

I guess I need a larger package like a logic-probe enclosure. That would lend itself to a bar-graph-style display.

But for now, I'll just roam the environment, listening for different sources of magnetic energy. ❑

Listing 1—*PicBasic code selects the proper gain for a* `best-fir` *input to the PicStic's onboard ADC.*

```
symbol    GAIN=B0
symbol    TEMPVAL=B2
symbol    BUTCNT=B5
symbol    ADVALUE=W10
symbol    GAIN10=0
symbol    GAIN100=1
symbol    GAIN1000=2
symbol    GAINA0=3
symbol    GAINA1=4
symbol    SPEAKER=5
symbol    CHANGEGAIN=6

SETUP:    peek $81,TEMPVAL
          TEMPVAL=TEMPVAL & $7F
          poke $81,TEMPVAL
          GAIN=1
          low GAIN10
          low GAIN100
          low GAIN1000
          low GAINA0
          low GAINA1
START:    call ADO
          TEMPVAL=ADVALUE/32
          sound SPEAKER,(TEMPVAL,8)
LOOP:     button CHANGEGAIN,0,10,50,BUTCNT,0,START
          if GAIN=1 then G1
          if GAIN=10 then G10
          if GAIN=100 then G100
G1000:    GAIN=1
          low GAIN10
          low GAIN100
          low GAIN1000
          low GAINA0
          low GAINA1
          got0 LOOP
G100:     GAIN=1000
          low GAIN10
          low GAIN100
          high GAIN1000
          high GAINA0
          high GAINA1
          got0 LOOP
G10:      GAIN=100
          low GAIN10
          high GAIN100
          low GAIN1000
          low GAINA0
          high GAINA1
          got0 LOOP
G1:       GAIN=10
          high GAIN10
          low GAIN100
          low GAIN1000
          high GAINA0
          low GAINA1
          got0 LOOP
```

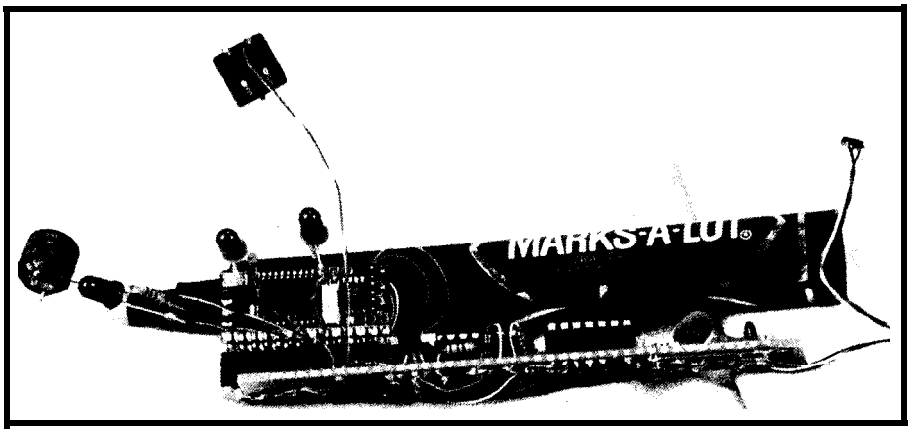


Photo 1—*Originally* slated for **insertion** *into* **this marker,** *the* **circuitry** *left* **no room** *for the* **batteries.** *Maybe the next prototype will use surface-mount technology. Notice the* **Hall-effect sensor dangling in** *the* **air on the** *far right.*

Jeff Bachiochi (pronounced "BAH-key-AH-key") is an electrical engineer on Circuit Cellar INKS *engineering* staff. *His background includes product design and manufacturing. He may be reached at jeff.bachiochi@circuitcellar.com.*

SOURCES

PGA204
Burr-Brown Corp.
6730 S. Tucson Blvd.
Tucson, AZ 85706
(520) 746-111
Fax: *(520) 889-1510*
www.burr-brown.com

HAL400S, LM336-2.5, ICL7660, LM660
Digi-Key Corp.
701 Brooks Ave. S
Thief Falls, MN 56701-0677
(218) 681-6674
Fax: (218) 681-3380

PicBasic Compiler
microEngineering Labs
P.O. Box 7532
Colorado Springs, CO 80933
(719) 520-5323
Fax: (719) 520-1867
info@melabs.com
www.melabs.com

PicStic-3
Micromint, Inc.
4 Park St.
Vernon, CT 06066
(860) 871-6170
Fax: (860) 872-2204
www.micromint.com

LM660, **LM336-2.5, ICL7660**
National Semiconductor
P.O. Box 58090
Santa Clara, CA 95052-8090
(408) 721-5000
Fax: (408) 739-9803

I R S

425 Very Useful
426 Moderately Useful
427 Not Useful

SILICON UPDATE

Tom Cantrell

Hot Chips IX

Just back from the Hot Chips show, Tom charts odd aberrations in memory, updates us on CPU extensions and processor/FPGA combinations, and snaps the latest picture of the digital-camera world.

Though you wouldn't know it from the earnings reports, Silicon Valley does seem to slow down a bit during the summer. I guess even the most caffeinated Java junkies need to take a break now and then.

To me, the Hot Chips show, usually around the end of August, marks the end of the lazy days. It provides a good opportunity to get the creative juices flowing as business activity heats up for the fall.

More importantly, in a business dominated day-to-day by technical minutiae, Hot Chips offers a good view of the forest. A good example is the tutorial *Sorting Out the New DRAMs* presented by Steven Przybylski (pronounced "shibiliski"), a well-known author and memory-IC guru.

RAM CRAM

If there's one thing I've learned during my travels with computers, it's that all the old saws about software expanding to fill all available memory are true.

Indeed, with the bloatware trend continuing unabated, increasing memory size turns out to be a surprisingly expedient and effective way to boost performance. Consider Figure 1, which benchmarks systems with various CPU and memory configurations. Notice, for example, that the performance of a 100-MHz Pentium system with 32 MB of RAM is higher than a 16-MB 200-MHz PentiumPro setup.

The good news is that the DRAM wizards continue to deliver ever-higher density, continuing the historic march from the original 4-Kb chips of the '70s. At this point, 64-Mb chips are moving towards crossover with 16-Mb chips and are poised for mainstream use in '98, especially as 32-MB PCs (i.e., four 8M x 8 DRAMs) become standard.

However, beyond 64 Mb, things get a little hazy. There's some speculation that a half-step move to 128 Mb could occur, although DRAMs historically hop by factors of four. However, 128 Mb makes sense in light of process and 12" wafer migration logistics.

Most interest is centered around 256-Mb chips, which are expected to be introduced before the turn of the century. However, unresolved issues at 256 Mb and beyond include the organization and interface. While DRAMs can keep up on the density front, speed (or rather, the lack of it) becomes the showstopper as processor megahertz outrun memory megahertz.

Until now, other than cramming in more bits, DRAMs haven't changed much. Over the years, the familiar RAS/CAS interface has persisted, with only the addition of fast page mode (FPM), which itself has been tuned up with the EDO (Extended Data Out) upgrade. The latter, involving a minor change in the function of the CAS line, delivers incremental speed improvement at essentially no cost.

However, the traditional DRAM design is finally pooping out, simply unable to keep pace with ever-faster CPUs. The near-term solution is Synchronous DRAMs (SDRAMs). As the name implies, SDRAMS (see "I Sync, Therefore I DRAM," *INK 55)* rely on dual banks, wide data paths (4-16 bits with 32 bits on the drawing board), and a high-speed (100+ MHz) clock to deliver data in a timely manner.

But not timely enough as it turns out, with even 100+ MHz SDRAMs just able to meet 66-MHz PC-bus timing specs, which allow for barely 9 ns from clock to data. To move forward, there's a proposal afoot for the so-called SDRAM-II, which offers a DDR

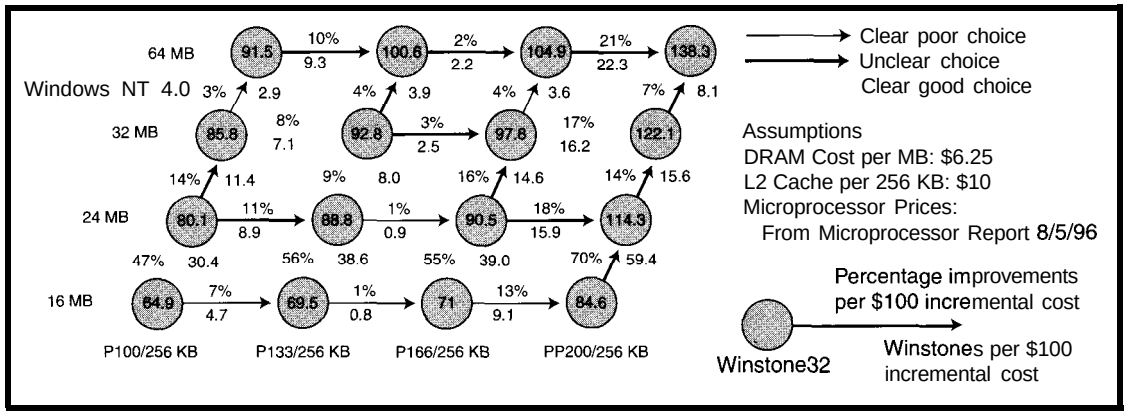


Figure 1—*Data* **prepared by** *Samsung,* **a top DRAM supplier,** *illustrates* **the CPU versus** *memory-size tradeoff Although the detailed recommendations may change over time (as relative CPU and memory* **prices fluctuate), the message that** *more memory is* **better comes through loud and clear.**

(Double Data Rate] feature by the simple expedient of transferring data on both clock phases as illustrated in Figure 2.

Although SDRAMs are shipping now, they face continuing technical and business challenges. Despite the fact that there's a JEDEC standard, subtle spec incompatibilities have plagued different manufacturers' chips.

Furthermore, while lower speed devices get by with TTL-like signalling, there's confusion surrounding the alphabet soup (GTL, CTT, TLVTTL, HSTL, SSTL, etc.) of contending higher speed interfaces. Same goes for SDRAM DIMMs, which are available in both 168- and 200-pin variants, with and without buffering.

These technical gotchas might work themselves out, but it's likely they won't have a chance in light of the recent decision by Intel to bless the Rambus RDRAM alternative.

First-generation RDRAMs achieved some impressive design wins-notably including the Nintendo 64, proving the viability of the chip and the business model. {Rambus doesn't make chips but instead licenses their designs to mainstream memory suppliers.)

The first-generation parts, thanks largely to a carefully tuned 1-V current mode interface, are speedy indeed, able to deliver 500-MBps peak bursts across a byte-wide channel.

Although details aren't known at this writing, Intel's spin on RDRAMs is likely to target even higher speed (how does 1.6 GBps sound?) by boosting the clock, doubling width to 16 and 18 bits, and enhancing the protocol's capability to handle multiple outstanding (i.e., split) transactions simultaneously.

However, not everyone is especially comfortable with Intel butting into the DRAM market. Their deal with Rambus isn't a simple bucks for know-how affair like everyone else's For instance, depending on exactly how things play out, Intel may start getting a piece of the RDRAM royalties or even a seat on the Rambus board.

Thus, it may be as much marketing as technology behind the SyncLink RAM, now known as SLDRAM. Backed by a number of heavyweight chip and system outfits (including most of the RDRAM suppliers), this effort proposes to reassert the traditional standards' (a la IEEE and JEDEC) approach to come up with an open Rambus alternative.

On paper at least (reference designs are supposedly underway), SLDRAM looks competitive. However, it remains to be seen just how Intel's RDRAM decision affects SLDRAM's acceptance.

SHADES OF CRAY

Of course, there was no shortage of big-ticket CPU chips disclosed. Take, for example, the 266-MHz Motorola MPC750 PowerPC or Sun's 300-MHz UltraSparc IIi. The former is a relatively lean and mean design (notably low power at only 5.5 W at 2.5 V), while the latter aggressively pursues floating-point performance.

Neither dethrones the Digital Alphas, but both are faster than a Pentium II (the Sun chip offering twice the floating-point performance).

Most of the CPU action these days centers on graphics, with debate focusing on whether the CPU or a special coprocessor should be in charge.

CPU extensions like the well-known MMX (Intel), VIS (Sun), MAX (HP), and the forthcoming MDMX (SGI) and MVI (DEC) all basically work the same way—namely, adding subword parallel operation capability. What this means is that a 32- or 64-bit register is treated as a vector of smaller (e.g., 8- or 16-bit) operands which can be added, multiplied, and otherwise crunched as a group.

Even discounting marketing hype, it's rather clear these schemes do offer a significant boost for various media-oriented applications. The Hitachi SH4 designers report a 4x speedup for 3D graphics, while the NEC V830 MIX2 extensions (56 vector instructions) easily cranks through MPEG2 decoding.

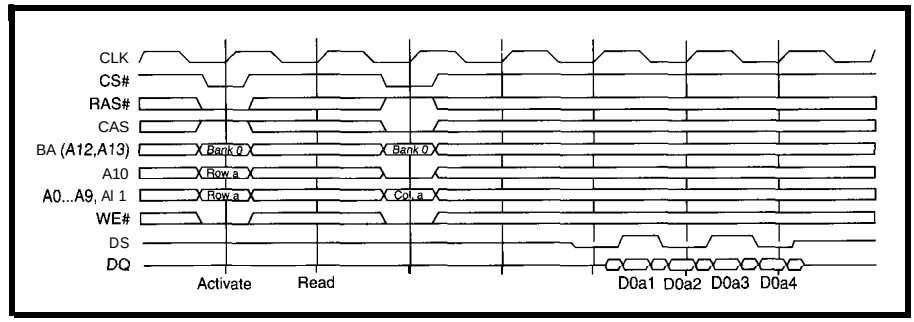


Figure 2—*The quest for* **speed demands transferring data on both clock edges in this proposed** *DDR (Double Data Rate) SDRAM. Note the DS (Data Strobe) signal that flows with the data (i.e., one DS for each SDRAM) to mitigate* clock *skew and* **variable** *flight-time* **problems.**

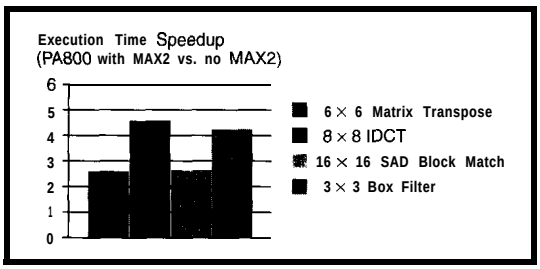


Figure 3—*Benchmarks from* HP highlight *the benefit of adding media-oriented vector instruction-set extensions. For fairness, the code was optimized for both cases, since even without the extensions, the PA8000 has useful media-processing features (e.g., bit field ops, multiply and accumulate, cache prefetching, etc.).*

HP reports similarly impressive results, as you see in Figure 3. These excellent results serve as a fitting testimonial to Seymour Cray who pioneered the vector computing underpinnings now at work on our screens.

The other school of thought is that all media processing, especially 3D, should be handled by dedicated chips, lest the CPU get bogged down shuffling all the bits.

An example of effective graphic coprocessing that's close to home is the Nintendo 64. Though perhaps not the commercial hit some expected, anyone who's seen one of these puppies in action knows the graphics performance is nothing to sneeze at.

And, it's all the more impressive given the price goals: $250 at introduction and ultimately heading toward $100 (admittedly, the business model relies on game royalties].

Meeting such price goals requires slashing chip count, but what few chips there are get the job done. An NEC 4300 CPU runs the show, certainly no slouch even running at "only" 93.75 MHz, combined with the aforementioned 500-MBps

However, it's the so-called Reality Coprocessor (RCP) that handles the gory details of graphics and audio. Inside the RCP, you can find the signal-processing subsystem shown in Figure 4, comprising a scalar control processor and an eight-element vector processor.

Interestingly, the scalar processor is a baby version of the main processor, using the same MIPS IV instruction set, though only for tiny subroutines that fit in the units' 4 KB each of instruction and data memory.

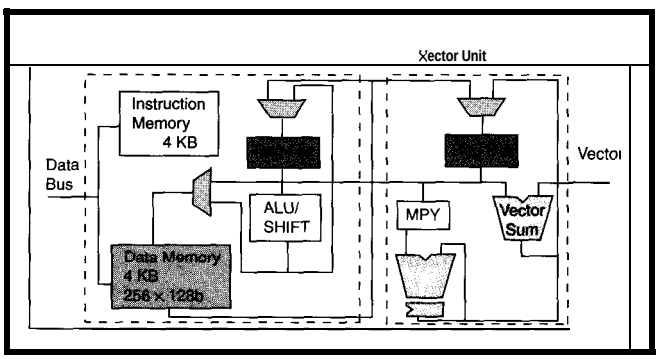


Figure *4-Processors proliferate inside the Nintendo 64, where the Reality Coprocessor (RCP) contains a signal processor comprising a 32-bit scalar processor coupled with an eight-element 16-bit vector processor.*

Meanwhile, the vector unit does the heavy lifting, able to perform up to 500 million multiplies and accumulates per second on vectors (eight 16-bit or four 32-bit operands) stored in thirty-two 128-bit registers.

FIELD-PROGRAMMABLE ANYTHING

The line between hardware and software continues to blur. There seems to be little difference between writing a program for a CPU in C or synthesizing gates in an FPGA using Verilog.

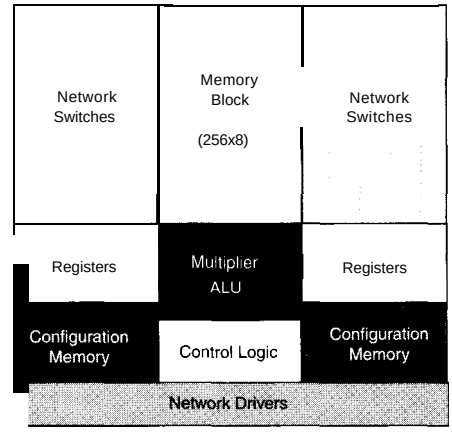
The line may disappear completely if chips like Matrix from the MIT Artificial Intelligence Lab take off. Noting that processors and FPGAs complement each other-the former suitable for control and coarse data, the latter best for regular fine-grained operations, Matrix combines the two concepts.

The chip includes a programmable interconnect of Basic Functional Units (36 BFUs in the first prototype), much like an FPGA. However, while the typical FPGA cell is limited to relatively simple logic operations, each Matrix BFU (see Figure 5) can be configured as a compute element or memory (either instruction or data). Using nearest-neighbor and length-four-bypass interconnects, arbitrarily application-specific machines can be constructed.

A demonstration of Matrix flexibility is shown by different implementations of a 16-bit integer FIR filter. The brute-force approach dedicates multipliers and adders, consuming up to four BFUs per tap but delivering results in a speedy two cycles.

Alternatively, a specialized FIR-oriented VLIW approach takes three cycles but consumes only 11 BFUs regardless of the number of taps. Finally, a microcoded implementation, much like a conventional CPU, saves space (8 BFUs) at the expense of more cycles (-8 x number of taps).

Figure 5—*Designers at MIT's AI Lab* **envision** *Matrix* **chips with hundreds** *of programmably* **interconnected Basic Functional Units** *(BFUs).* **Despite ifs name,** *the* **functionality of a** *BFU* **isn't basic** *at all,* **since it** *can be* **configured as practically anything from simple memory to** *a complete CPU.*



SMILE, YOU'RE ON CANDID CHIP?

For almost twenty years, CCD (Charge Coupled Devices) image sensors have ridden the home-video wave, migrated into scanners, and now are fueling the digital-camera craze.

However, the CCD solution is far from ideal, calling for a special mixed-voltage fabrication process and consuming a lot of power. Most notably, it's difficult to integrate any additional logic, making systems bigger, more expensive, and difficult to design.

That's all about to change with the emergence of CMOS image sensors. Until now, first-generation efforts have been limited to low-res single-transistor passive-pixel designs. However, thanks to ever-improving CMOS process density, investigation is centering on multitransistor active pixels that deliver excellent performance.

Consider the sensor, really a single-chip camera, disclosed by Photobit (see Figure 6). It features an impressive **512 x 384 x 8** pixel (each 7.9 x 7.9 μm) array and a bunch of other stuff, including the all-important ADC.

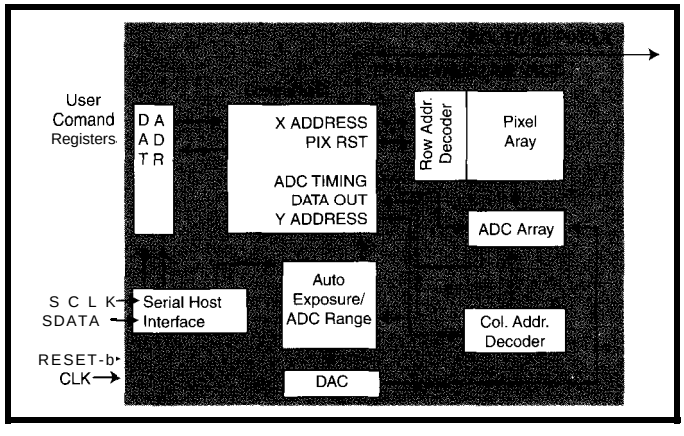


Figure B-Using CMOS instead of a CCD enables the integration of a complete single-chip camera which not only *cuts* **cost, size, and power** *but a/so eases system design by burying all analog processing on chip.*

In fact, the unit dedicates an ADC for each column to deliver image data at a speedy 14.3 MBps (i.e., 230 fps), yet only consumes -50 mW—a small fraction of the power demanded by a CCD.

There are more benefits to integration than simple downsizing. By incorporating the ADC, calibration chores can be handled on chip. And, the door is open for any and all manner of on-chip signal processing-filtering, gain, compression, special effects, and the like.

Heck, throw in a little digital audio, and tomorrow's smart camera might even spout helpful hints like "Lens cap, you doofus!" ❑

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. corn, by telephone at (510) 657-0264, or by fax at (510) 657-5441.*

REFERENCE

S. Przybylski, *Sorting Out the New DRAMs,* www.verdande.com.

SOURCE

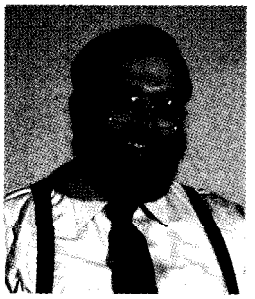
Hot Chips
IEEE Computer Soc.
701 Welch Rd., Ste. 2205
Palo Alto, CA 94304
(650) 941-6699
Fax: (650) 941-5048
www.hotchips.org

I R S

428 Very Useful
429 Moderately Useful
430 Not Useful

PRIORITY INTERRUPT

The Best Kept Secret



Recently, I went to the Embedded Systems Conference in San Jose. I've been going to the show since it started. When asked for my opinion of the show, "informative" is the term I generally use. After all, I've been going to shows since the microprocessor was invented, and "informative" is about as exciting as it gets. The notable difference this year was the marked increase in the number and stature of the exhibitors. I view that with mixed emotions.

For many years, embedded control has been a boring technical topic that everyone loves to ignore (except us, of course). Considered something only a bunch of gearhead engineers could understand or appreciate, the major technical media has focused instead on more visible computer applications like multimedia and smart networks. All of a sudden, it seems that we, or at least our down-in-the-dirt specialty, have been discovered.

For years, there has been a definite expansion in the embedded marketplace. For the most part, I feel it's been a steady and predictable evolution directed by the necessities of performance rather than any corporate master design. In fact, if anything, a total lack of regimentation coupled with an equal-opportunity mentality has enabled the industry to expand rapidly in many different directions at the same time.

The only principle universally applied in all embedded-design situations has typically been, "Does it solve the problem?" There are development-language preferences, but no absolute prejudices. There are platform and processor chip preferences, but no absolute architectures. There are price/performance goals, but no absolute cost thresholds.

My greatest fear is that we've been discovered! Being out of the limelight allowed us to design as engineers, not politicians. Consider desktops. What do you purchase for business today and how many people want to have something to say about it? You may be the best Mac expert in the company, but if management feels that Intel rules, your new desktop is a PC. When it's time to update software and the choice is between all Microsoft products or various selections from competing companies, does some IS manager 1000 miles away dictate conformity?

As a publisher, I welcome embedded control's new visibility. I can proudly sit back and say, "What took you so long?" and know that we were the pioneers. As an engineer, however, I suddenly wonder if all the new visibility will result in a self-conscious examination of our design techniques where none is required. Will embedded control development or architectures have to become politically correct?

Go ahead and laugh if you want, but this wouldn't be the first time sledgehammer electronics has been applied to simple control tasks. How many times have you thought that $50 worth of 16-bit real-time processing was a better alternative than even the least-expensive 32-bitter under Windows?

Sure, I'm comparing apples and bananas. But, that's not the point. What troubled me at the show was the sudden and massive presence of Microsoft and Intel (Wintel). Certainly, I'm wise enough to know a straight PIC or Z8 application isn't going to be replaced by an embedded PC. It's the middle ground where an engineer might use multiple 8-bit, a souped-up 16-bit, or a competing 32-bit processor that concerns me.

In this diverse, fragmented, and difficult to understand market, there hasn't been any pressure for an engineer to do anything except solve the problem. I'm just not sure what level of organization becomes too much. We have to be careful that Wintel predominance doesn't achieve for embedded control what it did for desktops-the virtual elimination of alternatives.

steve.ciarcia@circuitcellar.com