www.circuitcellar.com

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

#125 DECEMBER 2000

EMBEDDED DEVELOPMENT

Finding DSP Application Algorithms

A Controller-Based LCD Panel

Design2K Winner: The QuizWiz

Designing with SDRAM

ChipCenter: The Web's Definitive Electronics Resource

Address http://www.chipcenter.com/circuitcellar

ChipCenter

LOGIN/REGISTER

SITE NAVIGATOR

Buy Parts Now!

- myChipCenter
- Knowledge Centers
- Guides & Experts
- Product Reviews
- Communities
- Online Tools
- Circuit Cellar
- Resources
- Real Life

SEARCH CHIPCENTER

ASK US

THE ENGINEERS TECH-HELP RESOURCE

Let us help keep your project on track or simplify your design decision. Put your tough technical questions to the ASK US team.

The ASK US research staff of engineers has been assembled to share expertise with others. The forum is a place where engineers can congregate to get some tough questions answered, or just browse through the archived Q&As to broaden their own intelligence base.

★★★★★★★★★★

Test Your EQ

8 Additional Questions

CIRCUIT CELLAR ONLINE

THE MAGAZINE FOR COMPUTER APPLICATIONS

Circuit Cellar Online offers articles illustrating creative solutions and unique applications through complete projects, practical tutorials, and useful design techniques.

This Month | Archive | About Us | Contact | Looking for More?

THE ETHERNET DEVELOPMENT BOARD

by Fred Eady

Part 2: The Software and Firmware Exposed

Fred picks up where he left off and takes us through embedded Ethernet, as he explores part two of his online series. Looking at the Ethernet development board from a firmware angle, he covers a lot of information, but still reminds us, just as he does in all his print columns, that “It’s not complicated, it’s embedded.”

November 2000 GO

IR REMOTE-CONTROLLED VIDEO MULTIPLEXER

by Peter Gibbs

Costly technicians and intricate equipment doesn’t have to coincide with multi-channel video monitoring. Peter shows us a system for visual monitoring that can be designed so you can make simple modifications yourself, rather than having to call in that pricey expert. Using off-the-shelf components, Peter succeeded at his task, while not obliterating the budget.

November 2000 GO

A BETTER BATTERY CHARGER

by Thomas Richter

Part 2: Hardware and Software Implementation

Now that you’ve been introduced to the AVR Battery Charger Reference Design, Tom is ready to delve into the hardware and software. The ATtiny15 has special features that make it ideal for battery-charging applications. Take a look at the charge methods he details this time around.

November 2000 GO

STATE MACHINES

Learning the Ropes

by Ingo Cyliax

In this installment, Ingo illustrates state machines using algorithmic state machine charts, implementing them through one-hot state encoding, the implementation of choice for FPGAs. Ingo takes us through the basic design methodology, using a simple design with two states. Here, the hardware has to depend on internal state, instead of just computing an output value.

November 2000 GO

SCHEDULING REVISITED

Lessons From the Trenches

by George Martin

This month, George addresses the wide response he received for his past scheduling article. How can you schedule for something you’ve never done before? Well, keeping records is a good start. And, asking around to gather information isn’t a bad idea. Even if it comes down to taking a shot in the dark, George gives us some tips about breaking the schedule down.

November 2000 GO

INTERNET, OR ELSE!

Silicon Update Online

by Tom Cantrell

The Embedded Internet Conference has morphed from a workshop to a conference, occupying an even bigger space next year. Today’s designers are anticipating ’Net connection for all manner of electronics, from elaborate A/V remotes to LED gadgets galore. Everyone’s in hot pursuit, working under the premise that the answer will come later as to whether or not an Internet toaster is something people really want.

November 2000

Mfr Data Sheets

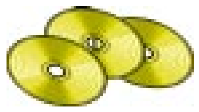
App Notes

Ask Us

SuperSearch

Search Type: Part Number

Search for: *lm317* GO



SAVE DOWNLOAD TIME

Order your own complete set of *Circuit Cellar Online* archives.

First 6 issues July-Dec 1999 are available.

CD Includes:

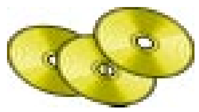
All HTML files

Article PDF

Code files

Navigate and use just as you would online.

(no Internet connection required)



RESOURCE LINKS

- **Solid State Relays**
- **Snubbers**

Bob Paddock

- **Flattened Displays**

Rick Prescott

INSIDE ISSUE 125

12 **SDRAM: The New Embedded Solution**
Mark Balch

20 **Make Your Data Comfortable**
Get Bit Cushions
Michael Smith & Laurence Turner

32 **Design2K Winner**
QuizWiz: A Hand-Held Scoring Device
Paul Kiedrowski

58 **Using a T6963 Controller-Based Graphics LCD Panel**
Brian Millier

68 **Designing for Reliability, Maintainability, and Safety**
Part 1: Getting Started
George Novacek

74 **From the Bench**
Sharing Technology with Mother Nature
Out of State with an Internet-Compatible Cell Phone
Jeff Bachiochi

78 **Silicon Update**
Hot Chips 12
Tom Cantrell

Task Manager 6
Rob Walker
How Will You Be
Paying for That?

New Product News 8
edited by Harv Weiner

Reader I/O 11

Test Your EQ 84

Advertiser's Index 95
January Preview

Priority Interrupt 96
Steve Ciarcia
Continuing The Plan

EMBEDDED PC

40 **Nouveau PC**
edited by Harv Weiner

43 RPC **Real-Time PCs**
A Cup of Java
Part 1: Embedded and Real-Time Applications
Ingo Cyliax

50 APC **Applied PCs**
Rabbit Season
Part 4: The Wonderful World of TCP/IP
Fred Eady

How Will You Be Paying for That?

Nothing in life is free. This is what my mother told me when I turned nine. She also informed me that I was old enough to get a paper route and was therefore capable of earning my own money, which was interpreted to mean, if I ever wanted to buy anything again, I had better call the *Hartford Courant* to see if they had an open route in the area.

The pride and honor of delivering the nation's oldest continuously-published newspaper faded with each falling snowflake that winter. But, I'm sure anyone who's ever delivered newspapers could share stories about trying to run away from a German Shepherd while wearing five layers of winter clothes, or the excitement and satisfaction that comes from riding through Mrs. NoTipper's tulip bed in the predawn darkness, so I'll move on.

The intended lesson was that if you want something, you have to work for it (the actual lesson went something like, delivering reading material to sleeping people helps you stop wanting things). In short, I learned to value the things I had to pay for.

When it comes to value and paying for things, one of the greatest debates of this year has been whether or not "available on the Internet" should be synonymous with "free." It's an interesting debate because there are good reasons to support both sides. Of course, the arguments can change depending on whether you're talking about music, books, videos, information, or services.

If I want information about refinishing furniture, I can find plenty of hobbyist sites with interesting (and sometimes useful) information or I can browse the shelves of the local Barnes & Noble. The problem with free stuff is, you can't complain when it doesn't work. Let's say I find a web site with information about a great new technique for refinishing furniture by burning off the old layers of paint. Who's to blame when my eyebrows and grandma's original Hitchcock chair are nothing more than a pile of smoldering ashes at my feet? For me, the value of credible information (that $20 book from B&N) is much higher than the possible "costs" of free information.

Besides, by the time you figure in the cost of your PC, hardware and software upgrades, the extra phone lines, ISP charges, and the time you spend searching the web and waiting for flashy ads (which pay for the information) to load, you'll see that "available on the Internet" comes with its costs. Don't get me wrong, there's more information on the Internet than in all of the local bookstores combined, but I think there are some things that are worth paying for.

Sure, I could log on to CNN.com or one of the local newspaper's web sites to get the news each morning, but waking up to the sound of a newspaper thumping against my door each morning gives me the satisfaction of knowing that I'm doing my part to help another young person learn that nothing in life is free.

Rob

rob.walker@circuitcellar.com

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

EDITORIAL DIRECTOR/PUBLISHER
Steve Ciarcia

MANAGING EDITOR
Rob Walker

TECHNICAL EDITORS
Jennifer Belmonte
Rachel Hill
Jennifer Huber

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
Mike Baptiste, Ingo Cyliax
Fred Eady, George Martin
George Novacek

NEW PRODUCTS EDITORS
Harv Weiner
Rick Prescott

PROJECT EDITORS
Steve Bedford, Bob Paddock
James Soussounis
David Tweed

ASSOCIATE PUBLISHER
Joyce Keil

CHIEF FINANCIAL OFFICER
Jeannette Ciarcia

CUSTOMER SERVICE
Elaine Johnston

ART DIRECTOR
KC Zienka

GRAPHIC DESIGNERS
Naomi Hoeger
Mary Turek

STAFF ENGINEERS
Jeff Bachiochi
John Gorsky

QUIZ MASTER
David Tweed

EDITORIAL ADVISORY BOARD
Ingo Cyliax
Norman Jackson
David Prutchi

Cover photograph Ron Meadows—Meadows Marketing
PRINTED IN THE UNITED STATES

ADVERTISING

ADVERTISING SALES REPRESENTATIVE
Kevin Dows — Fax: (860) 871-0411
(860) 872-3064 — E-mail: kevin.dows@circuitcellar.com

ADVERTISING COORDINATOR
Valerie Luster — Fax: (860) 871-0411
(860) 875-2199 — E-mail: val.luster@circuitcellar.com

ADVERTISING CLERK — Sally Collins

CONTACTING CIRCUIT CELLAR

SUBSCRIPTIONS:
INFORMATION: www.circuitcellar.com or subscribe@circuitcellar.com
TO SUBSCRIBE: (800) 269-6301, www.circuitcellar.com/subscribe.htm, or subscribe@circuitcellar.com
PROBLEMS: subscribe@circuitcellar.com

GENERAL INFORMATION:
TELEPHONE: (860) 875-2199 FAX: (860) 871-0411
INTERNET: info@circuitcellar.com, editor@circuitcellar.com, or www.circuitcellar.com
EDITORIAL OFFICES: Editor, Circuit Cellar, 4 Park St., Vernon, CT 06066

AUTHOR CONTACT:
E-MAIL: Author addresses (when available) included at the end of each article.

For information on authorized reprints of articles,
contact Jeannette Ciarcia (860) 875-2199 or e-mail jciarcia@circuitcellar.com.

CIRCUIT CELLAR®, THE MAGAZINE FOR COMPUTER APPLICATIONS (ISSN 1528-0608) and Circuit Cellar Online are published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (860) 875-2751. Periodical rates paid at Vernon, CT and additional offices. **One-year (12 issues) subscription rate USA and possessions $21.95, Canada/Mexico $31.95, all other countries $49.95. Two-year (24 issues) subscription rate USA and possessions $39.95, Canada/Mexico $55, all other countries $85.** All subscription orders payable in U.S. funds only via VISA, MasterCard, international postal money order, or check drawn on U.S. bank.

Direct subscription orders and subscription-related questions to Circuit Cellar Subscriptions, P.O. Box 5650, Hanover, NH 03755-5650 or call (800) 269-6301.

Postmaster: Send address changes to Circuit Cellar, Circulation Dept., P.O. Box 5650, Hanover, NH 03755-5650.

NEW PRODUCT NEWS

Edited by Harv Weiner

HIGH-VOLTAGE FLOATING AMPLIFIER

The **Model 237** DC coupled amplifier provides a frequency response from DC to 40 KHz (–3 dB at 40 kHz). Its input floats at ±20,000 V and its output is connected to ordinary lab instruments for data display and collection.

It has both real-time analog and digital outputs. Power on/off and gain is set with an ordinary infrared remote control unit (furnished) for hands-off operation.

The amplifier provides four gain ranges (x10, x1, x0.1 and x0.01) for input voltages of 0 to ±500 mV, ±5 V, ±50 V, and ±150 V (all peak-to-peak). The highest voltage range is limited by component withstand voltages. Input resistance is 1 MΩ on all ranges. The input can withstand ±150 V without damage whether the power is on or off

A real-time analog output of 0 to 5 V full scale (on highest three gain ranges) is available. An output of 0 to ±1.5 V is available on the x0.01 gain range. The low output impedance of the amplifier will drive voltmeters, oscilloscopes, analog recorders, etc. A 12-bit parallel, real-time digital output for 0 to 5 V full scale is available. This consists of two 8-bit bytes with the gain setting in the two most significant bits. This output connects to a computer parallel port with full handshaking.

The amplifier is housed in a plastic enclosure measuring 11″ × 7.75″ × 3″. It features BNC connectors for analog input and output, and a DB-25 connector for the digital output. Power is supplied from an included DC "wall wart" supply (the input amplifier board uses three AAA batteries).

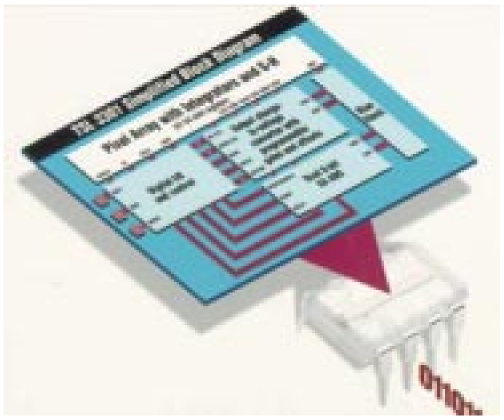
The Model 237 costs **$359**. A full datasheet and User Guide are available on the company's web site.

TDL Technology, Inc.
(505) 382-3173
Fax: (505) 382-8830
www.zianet.com/tdl

NEW PRODUCT NEWS

PIXEL ARRAY

The **TSL3301** Linear Optical Sensor Array combines a 300 dpi 102 × 1 pixel array with an 8-bit A/D converter and control circuitry. The monolithic IC operates down to 3 V and features a sleep mode for reduced power consumption. It is ideal for portable battery-operated applications such as hand-held scanners and OCR readers. Other applications include automotive steering control, robotics, linear and rotary encoders, and spectrometers.



It converts pixel data at a 1 µ/s. The array is split into three 34-pixel segments, each with its own set of programmable gain and offset registers. These registers provide the system designer with the capability to compensate for offset errors and to normalize the gain across the array.

The TSL3301 command set provides you with full control of the device. You can set pixel integration time, read and write to gain and offset registers, read data, and perform asynchronous pixel reset. The TSL3301 utilizes an easy-to-use serial interface for all control functions. Pixel output is serial, and each pixel value is represented by 8 bits. The serial data clock can run as high as 10 MHz for a 1 megapixel throughput rate.

The TSL3301 is supplied in an 8-pin clear epoxy DIP package with standard commercial temperature ratings (0° to 70°C). It is in an extended temperature glass-windowed package (–40° to 115°C) that is designated as the TSLW3301. The TSL3301 and TSLW3301 cost **$4.65** and **$5.58** respectively in 1000-piece quantities.

TAOS, Inc.
(972) 673-0759
Fax: (972) 943-0610
www.taosinc.com

READER I/O

ON TRACK

George, I really enjoyed your series "The Joys of Writing Software." (*Circuit Cellar* 121–123). It was thought provoking and is a must read for all of the software mongers out there. You're right on about how we expect hardware designers to follow rigorous design rules and practices in order to reduce costs of bad ASICs, new board spins, and missed market opportunities, but we neglect that this can and should also be done with software as well.

Some faculty I worked with in the past wrote a book titled *The Art of Hardware Design* (Prosser/Winkel), which explains that the art is more about applying proper design methodology to hardware design (e.g., using ASM charts (it was written in the '80s), control/data path decomposition, synthesizing state machines from it, following good synchronous design practice, and so on). Perhaps a similar book on software design would be a good idea. No doubt, you'll probably get your share of negative or cynical feedback from engineers who consider themselves to be "software artists." I just wanted to make sure you also get some positive feedback.

Ingo Cyliax

Thanks, Ingo. You're absolutely right about design process. My problem was finding a way to squeeze all I wanted to say into a manageable size for the articles. Hardware engineers also have a tendency to make the same mistake as software guys—they would like to draw schematics from the word go!

In my company, I insist that not a single circuit be drawn until the entire system has been defined in terms of functional blocks. It's difficult to enforce, but the price for cowboying is too high.

George Novacek

STILL LEARNING

Please do continue to keep us on your college-program mailing list. Now more than ever *Circuit Cellar* is welcome and continues to be viewed as the leader in computer applications. I believe that students want to continue (and that means buy) your magazine when they leave here.

I am now teaching introduction to microcontrollers and expect to add one or two advanced classes to meet the needs of our local technical and engineering students. I know how odd this may sound that at a two-year college we get students who already have their Bachelors in engineering, but I guess we are giving them something they can't get elsewhere. I love creating classes that bridge electronics, computers, and biomedical electronics together.

Please pass along how much this publication means to me and the students. Thank you.

Prof. William Schlick
Schoolcraft College
Livonia, MI

If you are a professor who would like to receive Circuit Cellar *to use and distribute in the classroom, please mail your request and e-mail address on college letterhead to:*

Valerie Luster
Circuit Cellar College Program
4 Park Street
Vernon, CT 06066

*Editor's note: The software for Edward Cheung's Internet project article (*Circuit Cellar *123) is moving from the URL listed at the end of the published article to www.cheung.place.cc.*

STATEMENT REQUIRED BY THE ACT OF AUGUST 12, 1970, TITLE 39, UNITED STATES CODE SHOWING THE OWNERSHIP, MANAGEMENT AND CIRCULATION OF CIRCUIT CELLAR, THE MAGAZINE FOR COMPUTER APPLICATIONS, 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, Steven Ciarcia, 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 is: A) Total number of copies printed (net press run) 31,559; B) Paid Circulation (1) Sales through dealers and carriers, street vendors and counter sales: 5,460, (2) Mail subscriptions: 18,061; C) Total paid circulation: 25,432; D) Free distribution by mail (samples, complimentary and other free): 854; E) Free distribution outside the mail (carrier, or other means): 4,905; F) Total free distribution: 5,759; G) Total Distribution: 31,191; H) Copies not distributed: (1) Office use leftover, unaccounted, spoiled after printing: 368; I) Total: 31,559. Percent paid and/or requested circulation: 80%. Actual number of copies of the single issue published nearest to filing date is (November 2000, Issue #124); A) Total number of copies printed (net press run) 31,300; B) Paid Circulation (1) Sales through dealers and carriers, street vendors and counter sales: 4,905, (2) Mail subscriptions: 18,946; C) Total paid circulation: 25,945; D) Free distribution by mail (samples, complimentary and other free): 800; E) Free distribution outside the mail (carrier, or other means): 4,450; F) Total free distribution: 5,250; G) Total Distribution: 31,195; H) Copies not distributed: (1) Office use leftover, unaccounted, spoiled after printing: 105; I) Total: 31,300. Percent paid and/or requested circulation: 83.2%. I certify that the statements made by me above are correct and complete. Joyce E. Keil, Associate Publisher.

FEATURE ARTICLE

Mark Balch

SDRAM: The New Embedded Solution

The next time you're designing a small, low-power embedded system, you might want to consider using an SDRAM controller. As Mark shows us, an embedded SDRAM controller just might be the simplest and most cost-effective solution.

Embedded systems have come a long way since the venerable '8051 microcontroller was hooked up to a 2716 EPROM. Embedded controllers today cover a wide spectrum including 8-bit single chip computers and 32-bit PCs. Each application imposes a different set of requirements for the design. A modern embedded controller may run at 32 kHz or 500 MHz, it may have 16 bytes of SRAM or 256 MB of DRAM and a huge hard drive.

I have always been attracted to the lower end of this spectrum where small size, low power, and custom hardware are keys to solving an embedded control task. In many of these applications, little scratchpad memory is required and the 32–128 bytes of local SRAM on the MCU are adequate. On the other hand, some other applications require data logging and often a common SRAM chip external to the MCU fits the bill perfectly. But what about those situations when a large quantity of local RAM is required? Enter DRAM.

DRAM has been around for many years and is one of the most common ICs manufactured. It's ubiquitous, found in PCs, printers, and consumer electronics. DRAM's advantage is its high density of storage; it packs more volatile RAM onto a square millimeter of silicon than any other technology. In most conventional computing applications, this density is the bottom line that renders DRAM's disadvantages insignificant next to SRAM.

For its density, DRAM typically consumes more power than SRAM, has a more complex interface, and is slower. However, power consumption usually is not an overriding factor in a corded appliance. The increased complexity of DRAM's interface was addressed a long time ago, first by dedicated DRAM controllers and now by integrated controllers on microprocessors and support chips. Finally, multilevel cache architectures mitigate the slower access time of DRAM.

Many embedded design engineers stay away from DRAM because of its power consumption and added complexity. The good news is that newer DRAM with lower supply voltage and power-saving modes uses less power. And, some embedded processors contain internal DRAM controllers, and if not, rolling your own for certain microcontrollers is not difficult.

Before you run off and start drawing schematics, let me provide one piece of advice: don't design DRAM into your

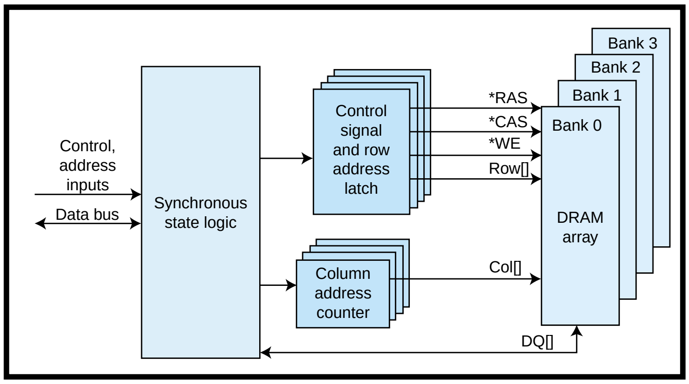


Figure 1—*SDRAM consists of a multibank conventional asynchronous DRAM array surrounded by synchronous state and interface logic. Separate enable logic is maintained for each bank, enabling simultaneous operations on multiple banks.*

embedded system! Ordinary DRAM is a thing of the past, and you don't want to get stuck designing with parts that are already obsolete. You want to use the new technology (e.g., synchronous DRAM (SDRAM)). SDRAM is easier to use than regular DRAM because it has a clean synchronous interface instead of an asynchronous enable architecture. Additionally, the benefits of riding the mass-market technology/economic wave of SDRAM are numerous. They range from widely available parts, to decreasing costs, to increasing speeds and densities.

WHAT'S SDRAM?

SDRAM is a new twist on the same DRAM technology that has been around for years. At the basic level, it can be thought of as a familiar asynchronous DRAM array surrounded by a synchronous interface on the same chip (see Figure 1). However, a key architectural feature of SDRAM ICs is multiple independent DRAM arrays—usually two or four banks. Multiple banks can be activated independently and their transactions interspersed with those of other banks on the IC's single interface. Rather than creating a bottleneck, this functionality allows better efficiency, and therefore higher bandwidth across the interface.

The synchronous interface and internal state logic enable these complex multibank operations and burst data transfers. After a transaction has been started, one data word flows into or out of the IC on every clock cycle. So, SDRAM running at 100 MHz has a theoretical 100-million-words-per-second peak bandwidth. In reality, this number is lower because of refresh and the overhead of beginning and terminating transactions. In fact, the true available bandwidth for a given application is dependent on that application's data transfer patterns and the SDRAM controller.

Rather than presenting a DRAM-style asynchronous interface to the memory controller, the SDRAM's internal state logic operates on discrete commands that are presented to it. There still are signals called RAS and CAS, but their functions are now different and they are sampled on a rising clock edge. The full set of control signals combine to form the standard SDRAM command set that the IC's internal state logic processes. These commands configure the SDRAM for certain interface characteristics such as default burst length, begin and terminate transactions, and perform regular refresh operations.

SDRAM provides high bandwidth in conventional computers because individual burst transfers require a fixed number of cycles of overhead to begin and finish. A cache controller, for example, might be able to fetch 256 words in only 260 cycles (98.5% efficiency). And that's just for starters. It could improve the overall efficiency by keeping track of its many transfers and interleaving them between multiple banks inside the SDRAM ICs.

This mode of operation allows a new burst transfer to be requested just prior to the current burst ending. Therefore, the overhead to start a transaction could be hidden within the time of another transaction. Theoretically, the SDRAM interface could be kept busy for every cycle, with the exception of the few cycles per second required to refresh the DRAM array.

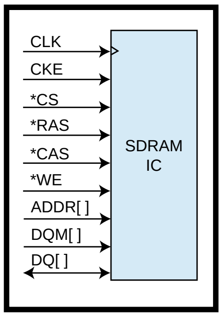


Figure 2—*SDRAM interface signals look similar to those of asynchronous DRAM, but the signals are sampled on the clock's rising edge and combine to form discrete commands.*

A SLEDGEHAMMER?

Don't worry if this sounds like a sledgehammer to hit your small, embedded nail. All this means is that the PC industry is supplying inexpensive, high-density, easy-to-use memory devices. Most of the whiz-bang SDRAM features are not useful for a slow microcontroller, therefore you can ignore them when designing an embedded SDRAM controller. In fact, for the basic data transfers, designing with SDRAM is practical and affordable. Just because these chips are meant to run faster than 133 MHz does not mean they can't run at 12 MHz.

If your embedded processor is heavy-duty enough to include an integrated SDRAM controller, you can stop reading, you're done! If not, you can make a simple controller to fit into a midsized programmable logic device (PLD). This controller must handle only the basics—a small subset of an SDRAM's full feature set. These tasks include powerup initialization, periodic refresh, and single-word read/write operations.

The main drawbacks of attaching an SDRAM controller to many small processors are setup latency and variable access time. At low frequencies, the setup latency is actually less than at the intended operational frequencies, but is still about three cycles for a read. The overall access time is variable because of the need for periodic refresh operations. Eventually a refresh will conflict with a memory access.

Command	*CS	*RAS	*CAS	*WE	Address	AP/A10
Bank activate	L	L	H	H	Bank, Row	A10
Read	L	H	L	H	Column	L
Read with auto-pre-charge	L	H	L	H	Column	H
Write	L	H	L	L	Column	L
Write with auto-pre-charge	L	H	L	L	Column	H
No operation	L	H	H	H	X	X
Burst stop	L	H	H	L	X	X
Bank pre-charge	L	L	H	L	X	L
Pre-charge all banks	L	L	H	L	X	H
Mode register set	L	L	L	L	Config	Config
Auto-refresh	L	L	L	H	X	X
Device deselect	H	X	X	X	X	

Table 1—*Some common SDRAM commands (H is logic high, L is logic low, X is don't care) are formed by combinations of control signals and samples on the rising clock edge. The AP/A10 signal contains valid address information for bank activate and mode register set commands.*

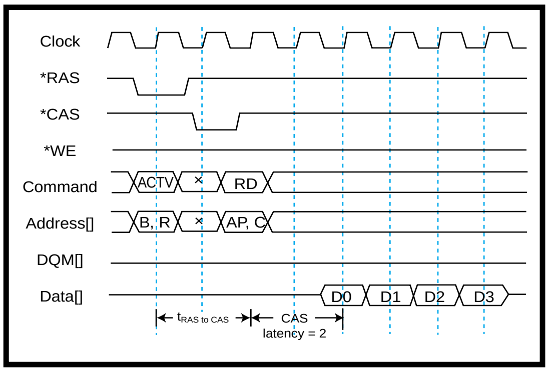


Figure 3—*A typical SDRAM read transaction consists of activating the desired bank, issuing the read command, and then waiting a specific number of cycles for the data burst to begin.*

When this happens and the refresh can no longer be held off, the processor will have to wait several cycles longer for the transaction to complete.

These drawbacks are not problems when dealing with processors that have acknowledge-terminated bus cycles. Such processors simply request a data transfer and then wait for an acknowledge signal to complete the access. But, most 8-bit processors do not have such advanced bus interfaces. This does not mean that the game is lost. Depending on the processor and application, there are a variety of clever solutions, which I'll cover later.

THE SDRAM INTERFACE

Before diving into designing an embedded SDRAM controller, let me explain the operation of the standard SDRAM interface. As mentioned, the memory controller uses a set of synchronous control signals to issue commands to SDRAM. Figure 2 illustrates the full set of interface signals, and the corresponding basic commands are listed in Table 1. The SDRAM samples the control signals on every rising clock edge and takes action as dictated. Some common functions include activating a row for future access, performing a read, and pre-charging a row (deactivating a row, often in preparation for activating a new row). For complete descriptions of SDRAM interface signals and operational characteristics, refer to any of the IC manufacturers' datasheets.

Figure 3 provides an example of how these signals are used to implement a transaction and serves as a useful vehicle for introducing the synchronous interface. *CS (not shown in Figure 3) is assumed to be tied low. The first requirement to read from SDRAM is to activate the desired row in the desired bank. This is done by asserting an activate (ACTV) command, which is simply asserting *RAS for one cycle while presenting the desired bank and row addresses.

The next command issued to continue the transaction is a read (RD). The controller must wait the necessary number of clock cycles that equate to the *RAS-to-*CAS delay time. This integer is different for each design because it is a function of the particular SDRAM's timing specification (in nanoseconds) and the clock period of your interface.

If, for example, your SDRAM's *RAS-to-*CAS delay is 20 ns and your clock period is 50 MHz or slower, you would be able to issue the RD command on the cycle immediately following the ACTV. Figure 3 shows an added cycle of delay, indicating a clock frequency between 50 and 100 MHz. During the idle cycles, the control signals are inactive.

The RD command is performed by asserting *CAS and presenting the desired column address along with the auto-pre-charge (AP) flag. The AP flag is conveyed by address bit 10 during certain commands such as reads or writes. Depending on the type of command, AP has a different meaning. In the case of a read or write, asserting AP tells the SDRAM to automatically pre-charge the activated row after the requested transaction completes. Pre-charging a row returns it to a quiescent state and also clears the way for another row in the same bank to be activated. Remember, a single DRAM bank can't have more than one bank active at any given time.

When would you want or not want to have the SDRAM automatically pre-charge the row after a transaction? This feature mainly serves to simplify those implementations that will not attempt to perform seamless, back-to-back transactions from the same row. If you are reading or writing just a few words to a row, having the SDRAM automatically pre-charge that row will spare your controller the effort of performing the pre-charge.

If your controller wants to take advantage of the SDRAM's seamless bursting capabilities, it may be worthwhile to let the controller decide when to pre-charge a row. This way, the controller can quickly access the same row again without having to issue a redundant ACTV command.

The AP flag also comes into play when issuing separate pre-charge commands. In this context, the AP flag determines if the SDRAM should pre-charge all of its banks or only the bank selected by the address bus.

Continuing the example, after the controller issues the RD command (called RDA if AP is asserted to enable auto-pre-charge), it must wait a predetermined number of clock cycles before the data appears. This is known as CAS latency in SDRAM jargon. SDRAM typically implements two latencies—two and three cycles.

Why would anyone select three cycles when two are faster? Like most things, you don't get anything for free. The SDRAM trades access time (clock to Q) for CAS latency. This becomes important at higher clock frequencies where access time is crucial to system operation. In these circumstances, you are willing to accept one cycle of added delay to achieve the highest clock frequency. This is because one cycle of delay will be

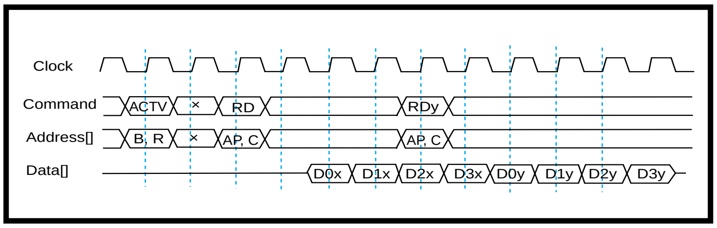


Figure 4—*Greater efficiency can be obtained by initiating a new transaction during the idle command time of a previous transaction. This results in seamless, back-to-back transfers that do not waste additional cycles of overhead.*

balanced by a higher burst transfer rate. Remember that SDRAM is powerful because of its superior bursting capabilities. At lower clock rates, however, it is possible to accept the slightly increased access time in favor of a shorter CAS latency.

When the CAS latency has passed, data begins to flow during every clock cycle. Data will flow for as long as the specified burst length. In this example, the standard burst length is four words. This parameter is configurable and adds to the flexibility of SDRAM. The controller can set certain parameters at startup, including CAS latency and burst length. The burst length defines the fundamental unit of data transfer across an SDRAM interface. Therefore, longer transactions must be built from back-to-back bursts and shorter transactions must be achieved by terminating a burst before it has completed. SDRAM enables the controller to configure the standard burst length as one, two, four, or eight words, or the entire row. It is also possible to configure a long burst length for reads and only single-word writes.

Writes, on the other hand, are not associated with latency. Write data begins to flow on the same cycle that the WR/WRA command is asserted.

The DQM[] bus provides one data/Q mask signal for each byte lane. So, an 8-bit SDRAM will have only one DQM signal, and a 32-bit device will contain DQM[3:0]. These mask signals enable or disable outputs during reads and enable/mask data during writes. DQM is useful for writes because it enables writing to fewer bytes than the data bus supports.

DQM assertion follows the timing of the read or write command. This means that the DQM for the first word of a transaction always will be asserted during the same cycle that the read or write is asserted. In the case of a read, the DQM leads the data by CAS latency cycles. In the case of a write, the DQM lines up with the data because writes do not have an associated latency.

After the transaction has completed, the row will either be left activated or pre-charged depending on the previous contents of the AP flag. If left activated, the controller may immediately issue a new RD or WR command to the same row. Alternatively, the row may be explicitly pre-charged. If automatically pre-charged, a new row in that bank may be activated in preparation for other transactions.

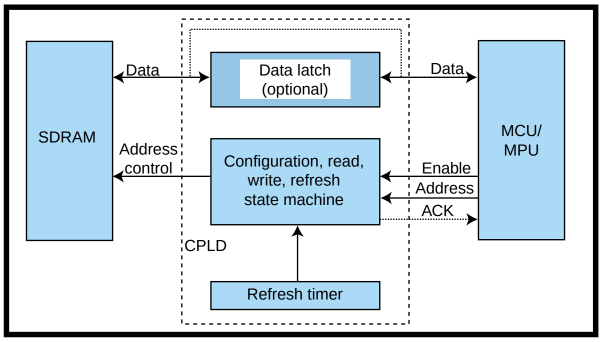


Figure 5—*A simple SDRAM controller for an embedded system can be designed into a modest CPLD. The basic elements are a command processing state machine, a refresh timer, and an optional data latch for certain types of implementations.*

SEAMLESS TRANSACTIONS

Thus far, I've alluded to the capabilities and benefits of seamless back-to-back transactions (see Figure 4). Because the command bus is essentially idle during the execution of a transaction request, new requests may be asserted prior to the completion of an in-progress transaction. The controller asserts a new read command for the row that was previously activated. By asserting this command two cycles (CAS latency) before the end of the current transaction, the controller guarantees that there will be no idle time on the data bus between transfers.

This concept can be extended to the general case of multiple active banks. Just as the controller is able to assert a new RD, it could also assert an ACTV to activate a different bank. Therefore, any of the SDRAM IC's banks can be asserted independently during the idle command time of an in-progress transaction. As these transactions near their ends, the previously activated banks can be seamlessly read or written to in the same manner shown in Figure 4. This provides a performance boost and can eliminate nearly all non-refresh overhead in an SDRAM interface.

SDRAM MAINTENANCE

As I stated, there are some operations that must be performed on the SDRAM to keep it functioning properly, including interface configuration and periodic refresh. At powerup (or at any other time), the memory controller is able to configure various parameters of the synchronous interface. These include the aforementioned CAS latency, burst length parameters, as well as a sequential or interleave burst mode setting. The controller configures these parameters by asserting a mode register set (MRS) command and encoding the parameters onto the address bus. At powerup, the controller must establish a known, stable state inside the SDRAM by executing several pre-charge and refresh commands prior to the MRS.

Periodic refresh is a universal requirement of standard DRAM devices, and SDRAMs are no exception. The basic concept behind DRAM refresh is that the tiny capacitors that store the state of each bit have a finite leakage rate. If left alone, they would lose their charge and the memory would forget its contents.

But, if the capacitors are refreshed before they can lose their charge, they will retain their state; hence, DRAM refresh. DRAM is refreshed one row at a time by feeding the outputs of the on-chip row sense amplifiers back into the same row. If the capacitors are able to reliably hold their charge for N s and there are M rows in the DRAM, you must refresh the array no less than M times every N s. Modern DRAM contains internal row counters that increment with every refresh cycle. This ensures that if M refresh cycles are executed every N s, each row will have its chance to be properly refreshed.

In reality, typical refresh periods are in milliseconds. A modern SDRAM may contain 4096 rows per bank, and require that all rows be refreshed every 64 ms. The controller must ensure that the rate is accomplished. They could be evenly spaced every 15.625 µs, or the controller might wait until a certain event passes and then rapidly count out the commands.

Different types of SDRAMs have differing refresh requirements, but the means of executing refresh are standard. All banks are pre-charged because the refresh (REF) command operates on all banks at once. When this is done, an executed REF command takes advantage of the SDRAM's internal refresh counter. The counter increments through each row every time the REF command is executed.

DON'T FORGET TIMING

It's easy to forget the asynchronous timing requirements of the DRAM core when designing around the SDRAM's synchronous interface. After studying state transition tables and command sets, the idea that an asynchronous element is lurking can become an elusive memory. Make sure that you double-check your integer clock multiples versus the nanosecond timing specs that are included in your SDRAM datasheet.

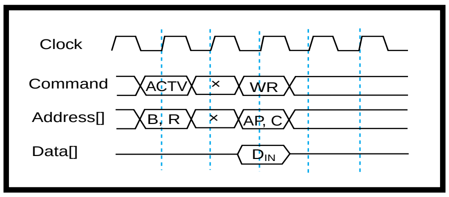


Figure 6—*The simplest read transaction consists of a bank activation, an auto-pre-charge read command, and a single word of outgoing data. This transaction is atomic because it does not rely on the state of a previous transaction and leaves the SDRAM in a "clean" pre-charged state.*

Examples include minimum and maximum RAS active time, RAS precharge time, and write recovery time. The tricky side of these timing specifications is that they affect your system differently depending on your operating frequency. At 25 MHz, a 20-ns time delay is less than one cycle. The delay is two cycles at 100 MHz. Errors like these may manifest themselves as intermittent data corruption.

IT'S EASY

You're probably wondering how this information is useful to a small, embedded system that isn't running a

32-bit microprocessor at 100 MHz. As I stated, you can pick and choose from what SDRAM has to offer. Take advantage of the high memory density and run at speeds of only a few megahertz to match your microcontroller. If 32-bit memories do not fit your application, you can use 8-bit devices.

The key to incorporating SDRAM into your embedded system is providing a simple memory controller to take care of the SDRAM housekeeping functions. The minimal set of SDRAM control logic can fit into modest PLDs. Some basic features are powerup configuration, periodic refresh, single-word read, and single-word write (see Figure 5).

That's it! All you need is control logic that can perform these four operations and you have a minimal SDRAM interface to your controller. There are as many ways to implement this basic idea as there are engineers. One technique is a single binary-encoded state machine inside a PLD. This state machine would be large, but if your system is only running at 12 MHz and you use a modern, mainstream PLD, the timing may be on your side.

HOW WILL THIS WORK?

The basic idea behind this simple SDRAM controller architecture is that all transactions are single words and they all execute atomically. That is, each read/write is automatically pre-charged so subsequent transactions do not have to take into account the previous state of the device; the SDRAM's banks will spend their idle time in pre-charge.

The logic to accomplish an atomic read transaction is simple: activate the bank/row, wait for the RAS-to-CAS delay (if necessary), issue the RDA, wait one cycle, and then latch the data on the next rising clock edge (see Figure 6). Assume the SDRAM was configured with a CAS latency of two cycles and a read burst length of one word. Depending on the operating frequency, you may be able to skip the single-cycle delay between ACTV and RDA.

A one-word atomic write is even easier: issue a WRA along with data after the RAS-to-CAS delay and return (see Figure 7).

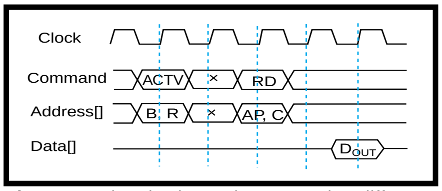


Figure 7—*The simplest write transaction differs from the read case only in that incoming data is presented in tandem with the auto-pre-charge write command. This transaction is also atomic.*

Implementing the periodic refresh can be done with a counter that rolls over no slower than the regular refresh interval specified by your device's datasheet. At each rollover, the state machine can simply issue the REF command, insert the appropriate delay, and then return to execute the next microprocessor request. The power-up MRS command will add several states to your state machine, but only has to execute once.

The only thorny issue related to connecting an SDRAM controller to a common 8-bit microprocessor is the lack of a request/acknowledge bus architecture. Such processors can't be held off when it takes longer than expected to provide data requested on a read transaction. The SDRAM's periodic refresh creates the problem that some transactions will take longer than others. In processors with an ACK signal, this is not a problem because the ACK would simply be delayed several cycles. But, if the controller can't hold off the processor, the processor will return to its software with false data. Solutions differ depending on the processor and circumstances.

In some situations, it may be valid to run the SDRAM at a higher multiple of the processor clock such that even worst-case transactions will complete in time for the processor's data valid window. Alternatively, some processors can be held off by stretching their clocks at the appropriate time during their bus cycles.

Inserting a dummy transaction prior to the processor's true data fetch operation is a third method. The processor tells the controller that it wants to read data. Then, the controller executes the read and holds the data.

The processor issues a second read request where it latches the read data. This usually works because there's ample time to execute a single-word read in the time it takes a slow processor to issue successive transactions.

In the case of a write, the processor could issue the write request along with data. The controller would then be able to latch that data and hold it until the WRA command is issued. This requires that the processor pause between successive writes to allow time for the operation. As with reads, this delay should not be more than one NOP or equivalent cycle.

IS SDRAM RIGHT FOR YOU?

Whether or not SDRAM is useful for your controller application depends on a variety of factors. Do you need more memory than can be easily accommodated by conventional SRAM? Will your controller board contain a PLD that is large enough to accommodate an SDRAM controller?

I hope you now agree it's a good idea to keep SDRAM in mind when solving controller design problems that require substantial quantities of local memory. SDRAMs are designed for high performance, leading edge systems, yet they may be applied to smaller and slower systems without excessive cost or complexity. This may enable you to design your way out of a memory tight spot in the future. ■

Mark Balch is an electrical engineer who designs FPGAs and PCBs in the data networking industry. He previously worked in the digital TV field where he participated in the development of various MPEG-2 broadcast products including an HDTV encoder. Mark earned a B.S. in Electrical Engineering from The Cooper Union. You can reach him at mark_balch@hotmail.com.

SOURCES

SDRAM ICs
Fujitsu Ltd.
(81-41) 754-3763
Fax: (81-41) 754-3329
www.fujitsu.com

Toshiba America, Inc.
(212) 596-0060
Fax: (212) 593-3875
www.fujitsu.com

FEATURE ARTICLE

Michael Smith
& Laurence Turner

Make Your Data Comfortable

Get Bit Cushions

What does it take to determine if your algorithm matches the characteristics of the processor you want to use? Tune in while Michael and Laurence illustrate the best ways for implementing algorithms on both integer and floating-point DSPs.

New floating-point DSP chips are hitting the market in the $5 to $10 range. Analog Devices came out with the ADSP21065L general-purpose 32-bit DSP processor with on-chip dual-ported memory and an independent I/O processor. When the algorithmic wind blows from the right direction, the '065L is capable of 180 MFLOPS (millions of floating-point operations per second). Texas Instruments recently announced the 120 MFLOP TMSV33. For a little more money, there is the quad-SHARC or TI's 150-MHz TMS320C6711 for even faster performance. If these new floating-point processors are so great, why do the same manufacturers still produce integer DSP processors in such large quantities? Analog Devices has its new 2.5-V ADSP2188 to compete against TI's TMS320VC5410.

In this article, we are going to take a design-oriented look at implementing algorithms on integer and floating-point DSP processors. How can you determine whether or not your algorithm will match the characteristics of a specific processor? Using integer processors means that you have to become more aware of the finite and quantized nature of the numbers you are handling. How do you determine the number of guard bits necessary to cushion your data against overflow or loss of precision? Are there equivalent problems and solutions when using floating-point processors?

BASIC DSP ALGORITHMS

Listeners perceive sounds as located inside their head when listening to prerecorded music on earphones (see Figure 1 [1]). An interesting context in which to look at the characteristics of DSP algorithms is to examine ways of altering these perceived sounds so that you can generate virtual speakers in front and behind the listener. This is one case when a code developer would be proud to announce, "Look, nothing between my ears!"

The simplest way of moving the perceived sound location is to delay the relative arrival of the sound from one ear to another. Imagine a single sound source placed 1 meter in front of your head. If your head is about 20-cm wide, the sound takes about 3.35 ms to arrive at each ear. Delay the sound by an additional 0.53 ms and 0.24 ms to the left and right ears respectively, and the position of the sound shifts 50 cm to the right. Listing 1 shows how to use a delay line to generate two independent sound sources from one, assuming sound sampling is occurring at 44 kHz.

There is only one problem, this doesn't work. Ears perceive relative, rather than absolute, delays. Although you might recognize two sounds in your headphones, it would be difficult to convince yourself that the sounds are not still positioned in the center and to the right of your head, along the line connecting your ears.

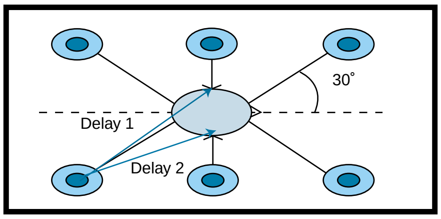


Figure 1—*The sound source is perceived to be in the center of the head if the same sound comes from both left and right earphones. If the sound is delayed before being sent to the left earphone, the perceived position of the sound source shifts to the right of the head. Further modeling of the audio channel using FIR and IIR filters can move the perceived sound out to a virtual speaker in front or behind the listener.*

To get a proper relocation of the sounds, you need a much more detailed algorithm capable of handling the concepts shown in Figure 2. [1] Here, we have taken into account the transfer functions of both direct and indirect sound arriving at your ears from the virtual speakers.

The direct transfer functions have to take into account that sounds constructively, or destructively, interfere after bouncing off of the earlobes or are blocked by facial anatomy. The degree of interference and blocking depends on sound frequency as well as sound location. The indirect transfer function accounts for multi-reflections and other characteristics of the room in which the sound is produced.

The direct transfer functions can be implemented by applying a real-time finite impulse filter (FIR) to the original sounds coming from the headphones. Listing 2 shows the C implementation of a FIR filter, which produces the output signal Output(n) as a weighted sum of the delayed input sound values Input(n).

$$\text{Output}(n) = \sum_{k=0}^{N-1} a_k \text{Input}(n-k)$$

In Listing 2, the storage of older input values is handled using a circular buffer implemented with pointer arithmetic. Some processors can handle this addressing more efficiently than the gross data movement of the delay line in Listing 1.

On more recent DSP processors such as the SHARC 2106X, circular buffer operations are implemented through hardware. This allows delayed values to be handled with zero overhead, which is a distinct advantage when trying to handle multiple FIR filters between audio samples.

Reverberation and other indirect sound effects can be achieved through the application of an infinite impulse response (IIR) filter:

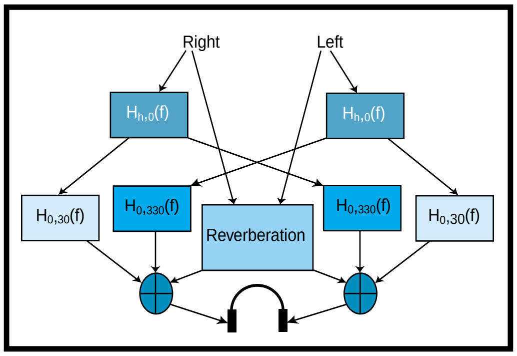


Figure 2—*The audio channel must be accurately modeled to make the sound from headphones be perceived as a set of left and right speakers positioned in front of the listener.*

$$\text{Output}(n) = \sum_{k=0}^{N-1} a_k \text{Input}(n-k) + \sum_{p=1}^{M-1} a_p \text{ Output}(n-p)$$

IIR filters make use of delayed versions of both the previous input and output values. In Listing 3, the C code of a basic IIR filter is implemented using a fast masking operation on an array offset to achieve circular buffer operations without the continual checking necessary in Listing 1. Masking, or modulus, operations on array indexes or pointers permit processors without specialized hardware to efficiently handle circular buffers.

Looking at the sound stage problem, there are eight major characteristics of DSP algorithms. First, they are memory- or register-intensive to store previous or intermediate results. Second, DSP memory address calculations compete for scarce processor resources. Third, DSP algorithms also typically involve multiplication.

Fourth, they involve lengthy summations where intermediate results can be large or small compared to the original input or final output values. And, DSPs involve loop operations whose checking can compete for processor resources. Next, DSPs need fast and easy access to peripherals such as A/D and D/A converters and timers.

Fancy, hardware-based, circular buffer (delay lines) and bit-reverse (FFT) addressing modes are characteristic of DSP CPUs. Lastly, there are no-overhead hardware loops.

Any processor, in principle, can handle DSP algorithms. However, real-time applications require a lot of processing between data samples. The custom features in a DSP-specific processor can prove invaluable. Fast

Listing 1—*C pseudo-code uses a delay line to generate a second perceived stereo sound source when only a single sound source is present. The delay line is implemented using gross data moves.*

```
#define MAXDELAY 1000
int delayline[MAXDELAY]

#define NORMAL_DELAY 147
#define LEFTEAR_DELAY 171
#define RIGHTEAR_DELAY 158

void MakeTwoSoundSources (int original_sound,
                          int *leftear, int *rightear) {
     int count;

// Place new sound into delay line
      delayline[MAXDELAY - 1] = original_sound;

// Pick up original sound source from delay line
      *left_ear = delayline[MAXDELAY - 1 - NORMAL_DELAY];
      *right_ear = delayline[MAXDELAY - 1 - NORMAL_DELAY];

// Pick up second sound source from delay line
     *left_ear += delayline[MAXDELAY - 1 - LEFTEAR_DELAY];
     *right_ear += delayline[MAXDELAY - 1 -RIGHTEAR_DELAY];

// Adjust volume
     *leftear = *leftear >> 1;
     *rightear = *rightear >> 1;

// Implement a very basic delay line operation
       for (count = 1; count < MAXDELAY; count++)
       delayline[count - 1] = delayline[count];
}
```

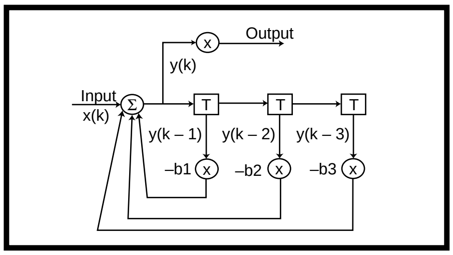


Figure 3—*Here's the direct form of an implementation of a third-order digital IIR Chebychev filter.*

access to intermediate or old values stored in memory is improved on the SHARC processor by on-chip memory.

Three data buses allow simultaneous access to two data memory blocks and an instruction cache to avoid data/data and instruction/data clashes. Two independent data address generators off-load address calculations from the main processor core.

However, all the fancy addressing modes and processing power available through the super-scalar processor can vanish into simple processed sand if one thing is not taken into account: Precise coefficient and data values are needed to achieve the specific desired DSP results.

SIGNAL QUANTIZATION

The signals and coefficients manipulated in a digital algorithm must be represented (stored) by finite, quantized, binary values. With integer processors, this process has two main consequences. At every point, or node, within the algorithm, there must be a finite range for the signals. There also is certain granularity in the signal. The granularity, or resolution, is the smallest value associated with changes in the least significant bit of the number representation. This is in the range of -2^{N-1} to $2^{N-1} - 1$, with a resolution of 1 for *2*'s complement numbers stored with a word length of *N* bits.

The quantization effects also are present with floating-point (FP) processors. The 32-bit IEEE single-precision FP format can store signals with maximum amplitude near 2^{127}. Although the smallest, nonzero magnitude signal that can be stored is about 2^{-127}, that is not the resolution available for FP numbers. The FP granularity changes from a fine resolution for small magnitude signals to a coarse resolution for large magnitude numbers.

This change is clear when you look at how FP numbers 178.125 and 0.6958008 are stored in memory. Table 1 shows how these decimal numbers are first converted to scientific decimal and then biased exponent binary numbers. Finally, the number is truncated to the 32 bits that can be stored in memory or register.

FP numbers are stored using 1 sign bit, 8 bits for the unsigned magnitude, biased exponent, and 24 bits for the signal magnitude. That's a neat trick with 33 bits stored in a 32-bit location! The signal magnitude is stored in the form of `1.factional_ part`. The *1* is "James Bond-ed"—implied, but not stored—in the number representation.

The granularity of the FP numbers is associated with changes in the last bit of the fractional part of the number representation. This granularity is effectively 2^{-23} times the value of the FP number that has been stored, so the resolution is continually changing. Changes near the floating point 0.695 can be recorded with 256 times greater accuracy than changes near 178.

COEFFICIENT PRECISION

All designers must become worried about the range and granularity of signals and coefficients imposed upon them by the finite word length associated with memory and data buses, registers, and processing units.

Listing 2—*C pseudo-code is necessary. The right ear sound is colored by the free-field transfer functions $H_{h,0}(f)$, $H_{0,30}(f)$, and $H_{0,300}(f)$. Here, the delay line is handled by a circular buffer implemented using software pointer arithmetic. Many DSP processors provide zero-overhead circular buffers through specialized hardware.*

```
#define MAXBUFFER 32
float circbuffer[MAXBUFFER]

// FIR coefficients - H_h,0(f) with H_0,30(f)
float FIR_H30[MAXFIR] = { …., …., …. }
// FIR coefficients - H_h,0(f) with H_0,330(f)
float FIR_H330[MAXFIR] = { …., …., …. }

void MakeTwoSources (int original_sound,
                      float *leftear, float *rightear) {
     int count;
      static float *new_valuept;
      float *pt = new_valuept;

// Generate FIR pointers to initial FIR coefficients
      float *ptFIR_H30 = &FIR_H30[MAXFIR - 1];
      float *ptFIR_H330 = &FIR_H330[MAXFIR - 1];

// Place new sound into circular buffer
      *pt   = (float) original_sound;

// Have the left ear hear the straight sound
      *left_ear = original_sound;

// Color right ear sound with both FIR filters
     *right_ear = 0;
      for (count = 0; count < MAXFIR; count++) {
       *right_ear += *pt * (*ptFIR_H30 + *ptFIR_H330);
              // Move to next filter coefficient
       ptFIR_H30- -;    *** NOTE --
       ptFIR_H330 - -; *** NOTE --
              // Move to next element in circular buffer
       pt- -;           *** NOTE --
       if (pt < circbuffer) pt = pt + MAXBUFFER;
      }

       // Prepare circular buffer for next new sound entry
      newvaluept++;
       if (newvaluept >= circbuffer + MAXBUFFER)
       newvaluept = nwevaluept - MAXBUFFER;
}
```

Figure 3 shows the direct form of an implementation of a third-order digital IIR Chebychev filter. The frequency response of this filter (see Figure 4) was designed for a passband finishing at 1 kHz and a stopband starting at 8 kHz. As Figure 4b illustrates, the maximum passband ripple is 0.1 dB. Achieving this filter response requires the following filter coefficients:

b_1 = 1.266 765 327 585 4
b_2 = –0.816 689 070 728 84
b_3 = 0.211 876 355 985 14

Unfortunately, whether using floating point, block-floating point, fixed point, or other number representations, the filter coefficients must be represented by a finite number of bits.

Let's assume for the moment that this third-order filter's coefficients must be quantized to an effective word length with 7-bit precision (6 bits plus a sign). Later, we will discuss what determines the effective precision of the coefficients.

With 7-bit precision, the best that can be done for the b_2 coefficient –0.816 689… is to implement it as the fraction 52⁄64, or 0.8125. The coefficient multiplication operation would be implemented as a multiplication of 52 followed by an arithmetic downshift by 6 bits (fast division by a factor of 64). As a result, the implemented transfer function of the filter is far from what was intended (see Figure 5).

Multi-precision operations may delay the onset of this quantization problem, but with loss of performance. However, a better solution is to use special filter design techniques. These manipulate all the filter coefficients so that, within the quantization limitations, the filter response is the closest possible to the desired filter response. This manipulation can be achieved by randomly hacking at the coefficients so that three wrongs nearly make a right. A more systematic approach is to use simulated annealing (i.e., equivalent optimization techniques).

Another technique is to change the form in which the algorithm is implemented. Figures 6a and b show the structure of a third-order Lossless Discrete Integrator (LDI) Chebychev filter and the associated passband detail respectively for 7-bit precision filter coefficients. [2] The transfer function for an LDI structure has reduced sensitivity to the values (word length) of the coefficients.

INTEGER OVERFLOW

Overflow can occur on integer processors when the algorithm involves repeated addition of numbers of the same sign. Implementations of FIR and IIR filters are subject to overflow. The overflow condition causes a result that is outside the range of the processor word length. The result is larger than the most positive number, or smaller than the most negative number that can be represented within the integer number representation.

If overflow is allowed to occur, severe signal distortion appears. For 2's complement, the effect of overflow at any node within the algorithm will

Listing 3—*C pseudo-code uses circular buffers so that the right ear sound is modified by an IIR filter. The circular buffers are implemented using masking operations on an offset rather than direct pointer arithmetic. This approach is useful on processors without hardware circular buffer capability.*

```
#define DATA_BUFFER   32
#define BUFFERMASK 0x1F
float in_buffer[DATA_BUFFER];
float out_buffer[DATA_BUFFER];

#define MAX_IIRCOEFFS 8
// IIR A coefficients
float IIR_A[MAX_IIRCOEFFS] = { …., …., …. }
// IIR B coefficients
float IIR_B[MAX_IIRCOEFFS] = { …., …., …. }

void MakeTwoSources (int original_sound, int *leftear,
                                            int *rightear) {
     int count;
        // Offset into circular buffer
     static int offset;
      float *in_pt = in_buffer;
      float *out_pt = out_buffer;

// Generate IIR pointers to initial IIR coefficients
      float *IIR_Apt = &IIR_A[MAX_IIRCOEFFS - 1];
      float *IIR_Bpt = &IIR_B[MAX_IIRCOEFFS - 1];

// Place new sound into circular buffer
      *(in_pt + offset)  = (float) original_sound;

// Have the left ear hear the straight sound
      *left_ear = original_sound;

// Color right ear sound with IIR filters
     *right_ear = 0;

       for (count = 0; count < MAX_IIRCOEFFS; count++) {
            *right_ear
                   += *(in_pt + (offset - count) & MASK) *
                  (*IIR_Apt);
            IIR_Apt --;
}

      for (count = 1; count < MAX_IIRCOEFFS; count++) {
        *right_ear
               += *(out_pt + (offset - count) & MASK) *
              *IIR_Bpt);
        IIR_Bpt--;
     }
      *out_pt = *right_ear;  // Store new output in buffer;
     offset = (offset + 1) & MASK;
}
```

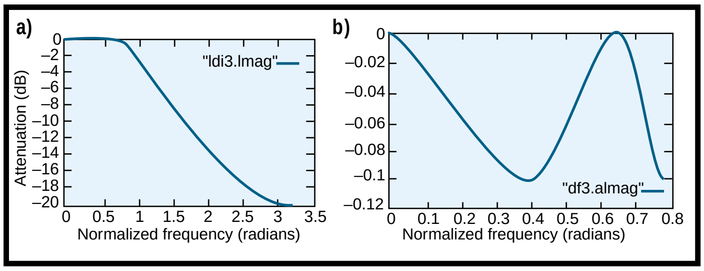


Figure 4—*This is a theoretical frequency response of a third-order direct form Chebychev filter. (**b**) is a zoomed version of the passband of the normalized filter response in (**a**).*

change the sign of the result. A severe, large amplitude, nonlinearity is introduced. To prevent this, the digital hardware must be capable of handling (representing) the largest number that can occur within the algorithm for any input signal.

Preventing overflow can be handled in two ways. If the word length associated with all internal nodes of the algorithm can be made sufficiently larger than the I/O signal word length, then all possible internal signal values can be accommodated and no overflow will occur. Some DSP algorithms can be efficiently performed using multiply and accumulate (MAC) instructions, which use an 80-bit accumulator for storing intermediate results.

If there is insufficient processor word length, then the input signal amplitude must be reduced so that no overflow is possible with any input signal. This reduction is obtained by scaling the input signal using shift operators rather than via implicit (slow) division. But, this scaling means a penalty in useful signal range. There is also an increase in the noise level at the output, because 1-bit inaccuracy at any stage in the algorithm has a greater effect on the scaled-down signal.

INTEGER OVERFLOW VERSUS FP RENORMALIZATION

Integer processors overflow when the intermediate results of an algorithm are larger than values that can be represented in the integer number representation. An equivalent effect can be seen in FP processors when the intermediate results are larger than can be represented in the fractional part of the FP representation.

When the FP result becomes too large, the processor renormalizes the number again. This means that the number associated with the fractional part still can be represented as `1.fractional_part`. But, the biased exponent of the FP number representation is adjusted to compensate. The larger number is still validly stored

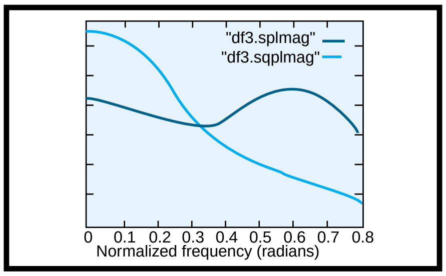


Figure 5—*Quantizing the multiplication coefficients to meet the available processor word length will generate band-pass characteristics that were not intended.*

within the FP representation, but with a reduced accuracy, because the granularity has changed.

Renormalization is a straightforward, but time-consuming operation. It is typically performed as part of operations using specialized fast, and/or highly pipelined, dedicated FP hardware. It will result in subtle nonlinear effects in most cases. However, if successive operations are performed on large and small FP numbers with differing signs, algorithm instability and inaccuracy can quickly appear.

ADVANTAGES OF FP AND INTEGER PROCESSORS

Until recently, one of the major advantages of integer processors was the speed of operations (particularly multiplication) and the lesser amount of silicon needed to implement the arithmetic units. Less silicon also means lower battery drain, which is important in small, standalone, embedded systems. Although this cost and performance effect is still there, changes in technology have decreased its importance.

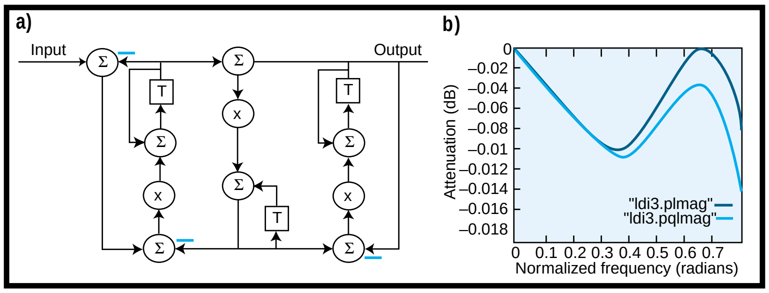
The major advantage of an FP processor is its automatic renormalization if the number to be represented becomes too large or, just as importantly, too small for the available word length. Renormalization allows the designer to be more flexible when handling worse-case scenarios of the interactions between algorithms and signals. This does not mean programmers can use FP processors without due diligence. Indeed, you need to know the fundamental limitations of both integer and floating-point number representations.

A useful feature in many integer processor architectures that partially compensates for the advantages of automatic FP renormalization is the sticky flag. This can play a useful role, as demonstrated in this DSP example of averaging 64 values of a noisy input signal that has been sampled using a 12-bit precision A/D converter:

$$\mathrm{Output}(n) = \frac{\sum_{k=1}^{64} \mathrm{Input}(n-k)}{64}$$

Although the average is guaranteed to be representable in 12 bits, it is possible for the sum generated to grow as large as 18 bits. Such a sum is not possible to accurately represent a 16-bit integer processor.

A number of things can be done to prevent



Figures 6a and b—*These show the structure of a third-order LDI Chebychev filter and associated band-pass detail, respectively, for 7-bit precision filter coefficients.*

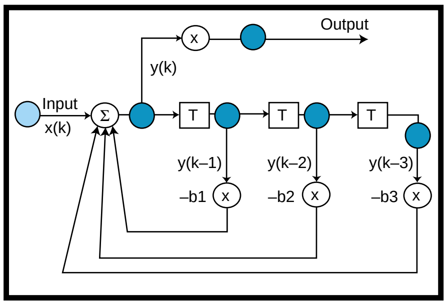


Figure 7—*The number of guard bits needed to protect against overflow can be calculated by knowing the maximum gain between the input and any internal node of the algorithm.*

the overflow of the sum. As suggested earlier, this problem can be overcome by prescaling the input. In this case, the input would need to be prescaled by four prior to calculating the sum:

$$\text{sum} = \sum_{k=1}^{64} \frac{\text{Input}(n-k)}{4}$$

Output(n) equals the sum divided by 16. The problem with this approach is that the output value loses precision as the input signal is effectively sampled with only 10-bit accuracy.

There is also an annoying side effect. The precision is lost even for input signals so small that no overflow would have occurred. To get around this problem, you need an algorithm that continually checks whether or not the sum is about to overflow and scales only when appropriate. This continual checking process places a strain on processor resources unless shifted to specialized hardware.

Many processors have sticky bits, which remember overflow conditions for many instructions. Rather than the slow checking for overflow at every individual stage in the algorithm, the sticky flag is set at the end. Then, if the sticky bit is set, the whole algorithm is repeated with scaled input values.

Other processors have accumulators with two or three times the standard word length. This guarantees that intermediate calculations of a sum can grow without loss of precision. But, if the algorithm requires storage of these wide intermediate values, then scaling must occur prior to storage with an associated loss of precision.

Many processors can be switched to employ saturation arithmetic. But, it can cause long-term inaccuracy, especially with recursive DSP algorithms such as IIR filters. Typical problems that can arise with saturation arithmetic are dead bands and limit cycles. Algorithms have dead bands when changes in the input are not reflected by expected changes in the output. Limit cycles are the opposite—changes occur in the output when there are no input changes.

An alternate approach with greater flexibility is to employ an FP number format. Here the numbers will be automatically scaled during the algorithm, only if necessary. This implies that FP operations allow for easier DSP algorithm design and provide results at least as accurate as an integer processor's results.

Although the first concept is true, the second idea can be misleading. Algorithm implementation in integer and floating point can't be discussed without specifying the processor! Imagine implementing the 64-point average using a 16-bit integer format and an imaginary processor capable of performing operations on the packed 16-bit FP format that is used for short word storage on the 2106 SHARC processors.

The 16-bit integer algorithm involves software scaling inputs by four, whether needed or not. The 16-bit FP algorithm requires no software scaling, because hardware renormalization will occur if needed. Keep in mind though, this 16-bit FP format uses 1 bit for sign, 4 bits to store a biased offset exponent, and 11 bits to store the rounded upper 11 bits of the source. With this format, even the addition of only two 12-bit input FP values has a better than 50% chance of causing renormalization and an associated loss of precision.

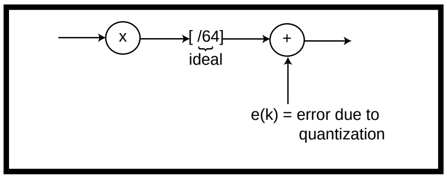


Figure 8—*Modeling signal quantization effects in terms of an ideal operation and an introduced quantization error signal is shown here.*

For the average algorithm with a 12-bit input, the 16-bit integer implementation will be more difficult to design, but gives better results. There just are not enough significant binary bits available in a 16-bit FP number representation. This is why FP processors have word lengths of 32 bits, to provide 24 bits to store the signal without significant loss of precision.

STAND ON GUARD

The design and speed of integer algorithms would significantly improve if you could guarantee that no overflow could occur and sufficient precision could be maintained, so that the output value is an accurate representation of the desired signal. This can be achieved by conditioning the input signal. Guard bits are placed above the sampled signal to prevent overflow. Other guard bits can be placed below the sampled signal to protect against loss of precision.

	Example 1	Example 2
Decimal value	178.1250	0.695808
Best integer approximation	178	1
Scientific decimal	1.78125×10^{2}	6.958005×10^{-1}
Scientific binary	$1.0110010001 \times 2^{111}$	$1.0110010001 \times 2^{-1}$
Biased exponent	$(1).0110010001 \times 2^{10000110}$	$(1).0110010001 \times 2^{01111110}$
Stored binary representation	01000011001100100010000000000000	00111111001100100010000000000000
Stored hexadecimal representation	0x43322000	0x2F322000

Table 1—*This shows the conversion of decimal values 178.125 and 0.6958008 (178.125 / 256) into their stored hexadecimal representation for 32-bit IEEE single-precision FP numbers.*

The analysis also provides design information useful when developing FP systems. First, the calculation gives an estimate of the dynamic range available before precision is reduced through renormalization. Equally important is that worse-case information is made available for deterministic timing calculations of subtasks involving real-time FP applications. Many pipelined FP processors take additional cycles if renormalization becomes necessary during an instruction.

For any algorithm, ask how many guard bits are needed. Also, be sure to ask where they should be placed in the processor word.

GUARDING AGAINST OVERFLOW

In principle, the determination of the optimum number of angel guard bits for any specific DSP algorithm is straightforward. You determine the maximum possible gain between the input step or node and any other step or node in the algorithm. Given this information, the input signal can be positioned on the data bus with a number of additional sign bits.

For the average example, the gain is simple to calculate (64), indicating the need for six additional sign bits beyond the input signal's word length to avoid overflow. Calculation of the necessary guard bits for the third-order digital IIR Chebychev filter (see Figure 3) is more complex because of the feedback. Such calculations are best automated using a tool such as DIGICAP. [3]

The maximum gain between the input and any internal nodes can be expressed in terms of the L1 Norms (see Figure 7). The L1 Norms were calculated using DIGICAP as:

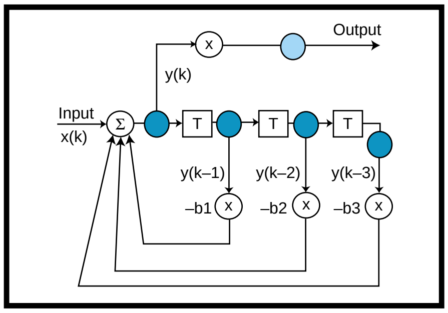


Figure 10—*Calculation of the number of guard bits needed to protect against quantization error requires determination of the gain between any internal node and the output.*

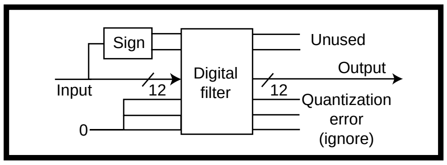


Figure 9—*Here you see positioning of a 12-bit input signal to avoid overflow and quantization errors for a third-order direct form digital Chebychev filter.*

L1(IN, OUT1) = 1.267
L1(IN, OUT2) = 1.267
L1(IN, DELAY1) = 3.6877
L1(IN, DELAY2) = 3.6853
L1(IN, DELAY3) = 3.6823

This calculation shows a maximum gain between the input and any internal node of 3.6877. Because this gain is less than 4 but more than 2, there is a need for two additional sign bits to protect against overflow for signals processed using this filter form.

GUARD BITS FOR PRECISION

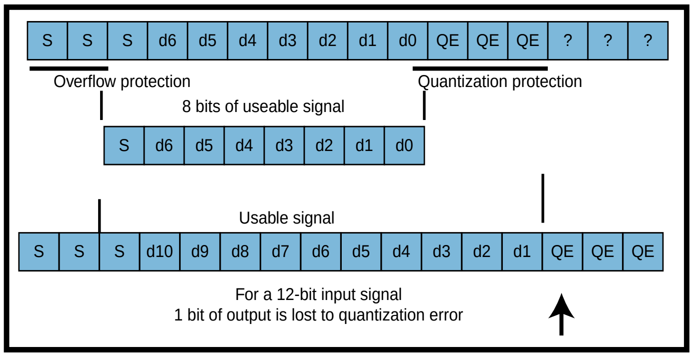
Because of the limited resolution of the digital signals used to implement the digital operations, it is not possible to represent the result of all operations exactly. Therefore, the signals in the filter must be quantized. The nonlinear effects caused by signal quantization can result in limit cycles. This means that the filter output may oscillate when the input is zero or a constant. In addition to that, the filter may exhibit amplitude dead bands where it does not respond to small changes in the input signal.

The effects of signal quantization are most obvious when coefficients are not exact integers. For example, the coefficient 0.8125, which would be implemented as an exact multiplication followed by a division and signal path quantization (truncation). One way to understand the effect of quantization is to treat the calculation of the result for every node in the algorithm as perfect, but introduce a small, random error (noise) generator at every node (see Figure 8).

The maximum amplitude of the noise generator is associated with changes of one in the least significant bit of the number representation. The maximum effect of this error noise signal at the output can also be determined if the L1 Norm (gain) between any internal node in the algorithm and the output are known. The calculation is shown in Figure 9.

The L1 Norms calculated for the third-order digital IIR Chebychev filter using DIGICAP are:

L1(OUT1, OUT2) = 1.0000
L1(DELAY1, OUT2) = 1.2657
L1(DELAY2, OUT2) = 1.2657
L1(DELAY3, OUT2) = 1.2657



S S S d6 d5 d4 d3 d2 d1 d0 QE QE QE ? ? ?
Overflow protection
Quantization protection
8 bits of useable signal
S d6 d5 d4 d3 d2 d1 d0
Usable signal
S S S d10 d9 d8 d7 d6 d5 d4 d3 d2 d1 QE QE QE
For a 12-bit input signal
1 bit of output is lost to quantization error

Figure 11—*For a third-order digital IIR Chebychev filter, a 16-bit integer processor provides an adequate word length to protect an 8-bit signal against overflow and quantization error. A 12-bit signal would lose 1 bit of precision in the output-to-quantization error.*

The values indicate that the maximum loss of precision between any internal node and the output is 1.26 bits. Also, it's possible for the quantization errors from one node to add to the quantization errors from another node. Therefore, the total possible quantization error is determined by summing the individual gains, giving 4.7973 bits. This indicates that three guard bits are needed to ensure that the least significant bit of the output has meaning.

EXAMPLE

You have seen that for an input signal of N bits, the third-order digital IIR Chebychev filter requires implementation with two guard bits to protect against overflow and three guard bits to protect against quantization errors. Figure 10 shows the ideal situation for a 12-bit input signal.

What do these calculations mean in real life when dealing with integer and floating-point processors? Figure 11 shows the ideal positioning of an 8-bit input signal within the word length of a 16-bit word processor. There are sufficient guard bits to protect against both overflow and quantization error. However, the 16-bit processor would not satisfactorily handle the third-order digital IIR Chebychev filter when the input signal is 12-bits wide.

The calculations for necessary guard bits to protect against overflow and quantization errors on integer processors can be reinterpreted for floating-point processors. Guard bits for overflow on integer processors translate into additional bits necessary to avoid possible loss of accuracy when floating-point numbers are renormalized. Guard bits that protect against quantization noise on integer processors effectively translate into the additional bits necessary to avoid floating-point truncation errors.

Figure 12 shows an 8-bit input value stored inside the 16-bit floating-point representation available on the SHARC processor. In this representation, there is 1 bit for the sign, 4 bits for the exponent, and only 11 bits to store the fractional part of the number. With two guard bits necessary to protect against normalization errors and three guard bits to protect against truncation errors, it appears this 16-bit FP representation is 1 bit short of accurately processing an 8-bit input signal through the Chebychev filter. However, the FP numbers are stored using a `1.frac` format, which makes an additional bit available.

Using an argument based on this analysis, a 16-bit floating-point processor has the precision of a 13-bit integer processor. And, a 32-bit floating-point processor essentially has the precision of a 25-bit integer processor.

In this article, we looked at a variety of techniques to ensure precision in DSP algorithms on both integer and floating-point processors. Both hardware and software features were discussed. The analysis tool DIGICAP is particularly useful, because it calculates the necessary guard bits to protect against overflow and quantization error. Copies of the DIGICAP disk with user manuals and examples can be obtained from Laurence Turner. ■

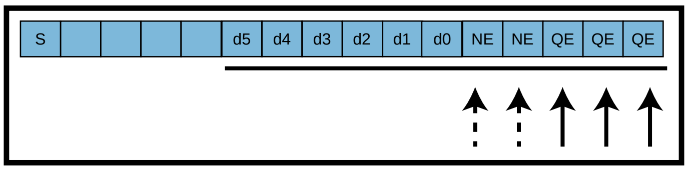


Figure 12—*For a third-order digital IIR Chebychev filter, a 16-bit floating processor provides an adequate word length to protect an 8-bit signal against floating-point normalization and truncation errors because of the hidden bit associated with the `1.frac` format of floating-point numbers.*

Mike Smith and Laurence Turner both are professors of electrical and computer engineering at the University of Calgary, Canada. Mike teaches and researches microprocessor operations with a biomedical slant. You may reach him at smith@enel.ucalgary.ca. Laurence teaches programming languages and ASIC design while researching digital signal processing system design and implementation. You may reach him at turner@enel.ucalgary.ca.

REFERENCES

[1] E. Bessinger, "Localization of Sound Using Headphones," Master's thesis, University of Calgary, 1994.

[2] L. Bruton, "Low Sensitivity Digital Ladder Filters," IEEE Transactions on Circuit and Systems, vol. CAS-22, March 1975.

[3] L.E. Turner, D.A. Graham, and P.B. Denyer, "The Analysis and Implementation of Digital Filters Using a Special-Purpose CAD Tool," IEEE Transactions on Education, Special Issue on Teaching and Research in Circuits and Systems, vol. CAS-32, no. 3, 1989, 287–297.

RESOURCE

M. R. Smith and L. E. Turner, "Finite Precision Effects in Digital Filter Implementations," SHARC '99 International DSP Conference, U.K., November 1999, 28–33.

FEATURE ARTICLE

Design2K Contest WINNER

Paul Kiedrowski

The QuizWiz

A Hand-Held Scoring Device

When it comes to making the grade, Paul's Design2K project passed the test with flying colors and won a grand prize. With the Quiz-Wiz, a teacher has a simple and cost-effective tool that can reduce the time spent grading quiz forms.

For automatic scoring of multiple choice tests, many schools use a commercially available system based on a desktop card reader machine, which requires that students mark their answers on preprinted forms of specific size and layout. This method is expensive because of the equipment and score cards, therefore, usually it's used only for critical testing.

In most cases, because only one centralized scoring machine is available to teachers, it is not located in the classroom where it would offer the most convenience. Perhaps more importantly, the most useful time to evaluate test results is immediately after a test so that feedback can be given and the missed questions discussed promptly. This is especially desirable for periodic quizzes where the intent is to allow the teacher to quickly gauge the classroom's learning progress. What is needed is a better, lower cost, convenient way of quickly scoring quizzes.

INTRODUCING THE QUIZWIZ

To answer these needs, I developed a hand-held scoring device based on a Philips 8-bit microprocessor. The 87LPC764 is a new 20-pin offering that combines fast speed, 8051 code compatibility, and low cost. This processor is ideally suited for the scoring device project because of its 4-KB code space, power-saving modes, serial port, and remaining 16 I/O pins. My objective was to create a device that is simple to operate and affordable enough that every teacher could own one (see Photo 1).

The QuizWiz has many features that make the teacher's ability to score multiple-choice quizzes fast and easy. It reduces scoring time to only 10 to 15 s per page. It is capable of scoring tests printed on standard paper without preprinted forms, using a word-processing template.

The QuizWiz does not require machine-assisted paper handling mechanisms. Also it operates on three AAA batteries. The device is inexpensive ($30 for parts) and measures only 6″ × 1.7″ × 1″. Simple to learn, the new teacher's aid requires only two buttons to operate.

A 2 × 8 LCD shows the status and results, and a buzzer provides distinctive audio feedback. Automatic power

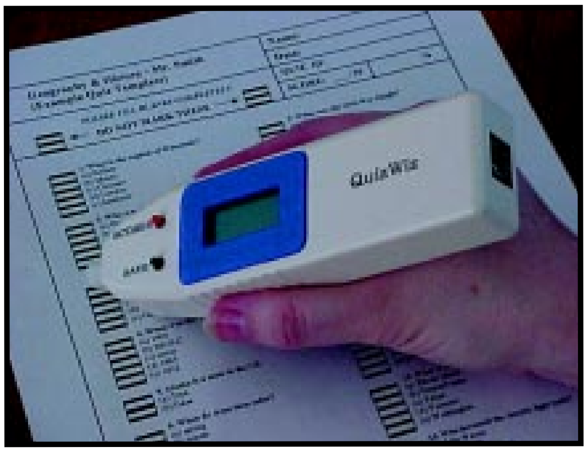


Photo 1—*The prototype QuizWiz sports a 2 × 8 character LCD display and just two operating switches labeled "scores" and "save." The quiz format can be seen here, requiring a starting sync section, dark areas between answer selections, and a minimum amount of white space between questions. A mini-DIN connector is on one end to provide an optional serial port interface.*

shutdown during idle periods provides long life. The flexible quiz format allows multiple columns and pages.

For convenience, there is temporary memory storage of results during power shutdowns. Totals, per question and per quiz, are available. The QuizWiz can handle eight choices per question, four columns of questions, and 32 questions per quiz. The processor provides an RS-232 serial connection for uploading results to a PC, which allows tracking of which questions were missed per student.

CIRCUIT DESCRIPTION

Figure 1 shows the circuitry has been partitioned by the two chassis sections. A PCB in the lower half contains most of the components, whereas the sensor tip, LCD, and battery compartment are in the upper half. The processor was DIP-socketed for the development phase (see Photo 2).

A single reflective optosensor was chosen to perform the scanning detection process, with an optimum sensor-to-paper distance of 1 mm. To preserve battery power, the optosensor LED is only activated when the QuizWiz is pressed against the paper, depressing a mechanical switch located in the tip. An alternate scheme that I initially considered required two sensors, the second one used for scanning a parallel column of markings intended only to synchronize the scan position. The additional LED would have significantly increased the power consumption, however.

Normal battery operating current is approximately 25 mA when all circuits are operating, 15 mA when not scanning, and 20 µA during shutdown. Using three AAA batteries in series with a typical capacity of 1000 mAh, the teacher can score approximately 100 quizzes for 30 students before replacing the batteries.

The QuizWiz uses a simple three-chip design consisting of the processor, a 5-V DC/DC converter, and an RS-232 three-wire interface. The 87LPC764 is a good match for the required features, because all of its pins and most of its features are used in this project. The only features not used are the I^{2}C interface and analog comparators. To minimize cost further, no external crystal is required, because the processor conveniently includes an internal 6-MHz RC oscillator.

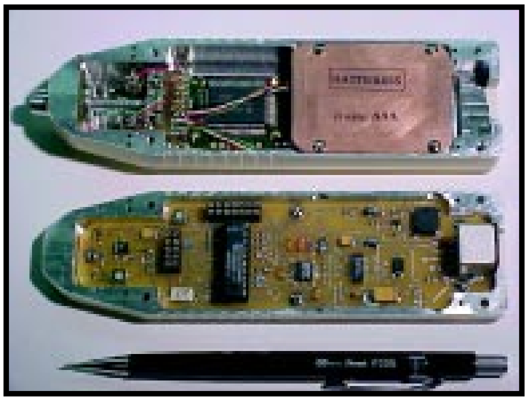


Photo 2—*A design-for-manufacturing approach was taken for the construction of the aluminum-chassis prototype. The main two-sided PCB has ample room for parts, mainly because of the housing size needed to fit the AAA battery pack and LCD display.*

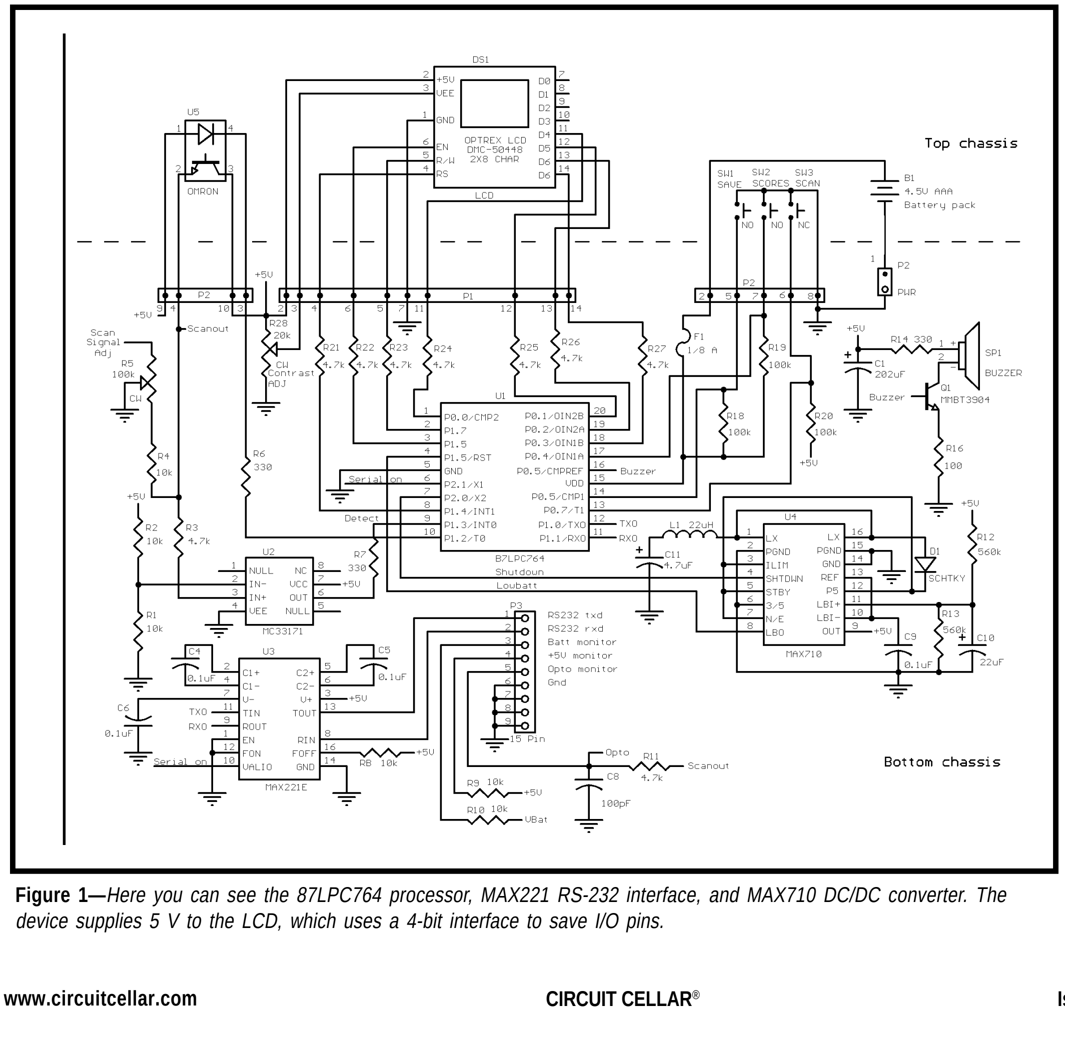


Figure 1—*Here you can see the 87LPC764 processor, MAX221 RS-232 interface, and MAX710 DC/DC converter. The device supplies 5 V to the LCD, which uses a 4-bit interface to save I/O pins.*

OPERATION

Photo 1 exemplifies how the quiz format is arranged to aid in the scanning process. The format requires questions to be arranged as usual in columns, with three additional constraints. These constraints include a short sync marking at the top of each column to aid in establishing the scan rate, a black or darkened area between each of the answer selections, and a minimum amount of white space between each question to distinguish them.

To use the device, teachers simply place the QuizWiz on the paper and slide it down along the check boxes used for multiple-choice selection by the students. The device scans the selections made and compares the results to the correct answers that were previously scanned in via a master quiz. After each quiz is scanned, the score is displayed as both the number of correct answers and a percentage. The teacher also can select the current cumulative scoring statistic for each question or the overall quiz, so the appropriate review focus can be established.

As stated earlier, the QuizWiz includes a sounding device to provide audio feedback. A short beep will be heard if a column is scanned properly, a longer beep when all columns of the quiz are correctly scanned, or various alarm beeps if an error is detected. After each successful quiz scan, the teacher presses Save or scans again.

At any time after a master quiz is scanned, the teacher can press the scores button for results of each question, as well as for other information. Simply continue to press the button to advance to the results for the next question. Note that holding down the scores button for at least 1 s resets the display and shows additional pertinent data.

Results are stored in memory unless battery power is removed or both the scores and save buttons are held down for more than 5 s. Then, the QuizWiz will erase all results and start over, directing the teacher that the next quiz scanned must be the master.

The QuizWiz goes into low-power shutdown mode after 2 min. of inactivity and is returned to normal operation when both scores and save are pressed together. A flow diagram of the operation is shown in Figure 2.

For access to quiz scoring details as they are scanned, you may connect a PC to the QuizWiz using a standard RS-232 serial port connection at 9600 bps. The QuizWiz automatically detects the presence of the serial port connection, and power usage is reduced when not connected.

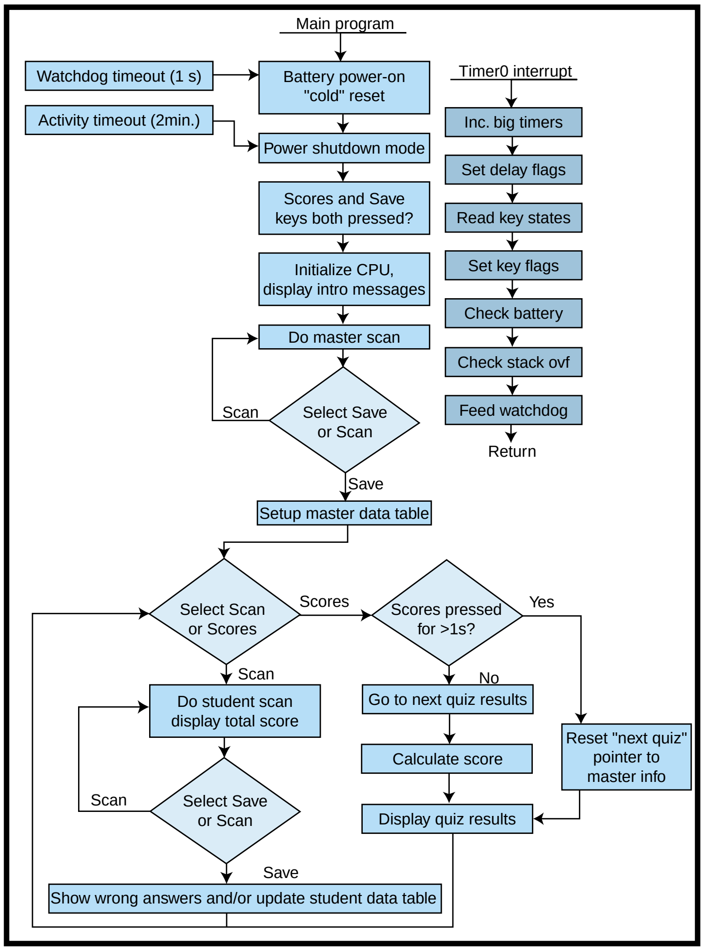


Figure 2—*In the operation flow diagram, notice that the scan switch is depressed whenever the tip is pressed to the paper. This approach minimizes power consumption by allowing the device to use only a single optosensor.*

ADDITIONAL FEATURES

There is no main switch needed to disconnect electronics from the battery, and existing data is preserved in RAM during shutdown. Although the processor has keyboard interrupt capability, a polling technique is used instead, because a main ISR timing loop is required. The circuit contains provisions to detect a low-battery condition and displays an appropriate message, then shuts down if necessary.

The device must not only correctly identify marked answers, but also be capable of reading multiple columns of questions and ensure that the total question count is the same as that of the master quiz.

Because the LCD used was chosen for its minimal cost and small size, it was important to carefully choose the formatting of the displayed results. For example, pressing the scores button presents the following sequence:

SCORING RESULTS:
TOTAL # QUES = XX
TOTAL # COLMN = X
TOTAL # QUIZ = XX
AVERAGE SCOR = XXX
QUES #01 SCOR = XXX
QUES #02 SCOR = XXX
(And so on, until all questions are listed.)
QUIZ #01, SCOR = XXX
QUIZ #02, SCOR = XXX
(And so on, until all quizzes are reported.)
NO MORE QUIZZES
(Then it starts again.)

HARDWARE ANALYSIS

The microprocessor has the capability to monitor keyboard interrupts, but cannot distinguish key states or releases. The keyboard interrupt is level sensitive and will continue generating interrupts until the key is released. Therefore, polling must be used to determine when multiple keys are pressed as well as for debouncing. For these reasons, it is simpler to use the Timer0 interrupt to monitor key states and actions, and only enable the keyboard interrupts for wake up from power shutdown. The nominal Timer0 interrupt rate is:

$$\frac{\left(\frac{6\ \text{MHz}}{6}\right)}{65536} = 15.26\ \text{Hz, or } 65.5\text{ms}$$

The behavior of the device during a scan is detailed in Figure 3. Assuming a 10″ column of 0.1″ markings per answer (dark plus light), scanned in 0.5 s, the minimum value of TSYNC is:

$$\frac{0.1}{10 \times 0.5} = 5\ \text{ms}\cdot$$

Assuming markings are .25 and the column scanned in 3 s, the maximum TSYNC is:

$$\frac{0.25}{10 \times 3} = 75\ \text{ms}\cdot$$

So, during scans, if you use Timer1 in 16-bit mode, it has a range of 0 to

65.5 µs, which is close enough for starters. You can improve the range by independently measuring TDARK and TLIGHT. To conserve memory usage, if you only store the upper eight bits of Timer1 when measuring, the precision will be 256 µs. That's good enough, because TDARK won't be less then 2 µs, and this error does not accumulate during the scan.

The Timer0 interrupt latency is not critical because the fastest operation you are trying to measure is TDARK to within 1/10 of its period, or approximately:

$$0.1 \times \left(\frac{5\text{ms}}{2}\right) = 250\mu\text{s}.$$

Therefore, you can either disable timer0 interrupts during this critical time (with an insignificant effect) or try to keep the timer0 ISR under 250 µs. If enabled, Timer1 should be set to a higher priority than Timer0.

The "big timers" are all 8-bit counts (reset at different times) of Timer0 interrupts, so the maximum range is 255 × 65.5 ms = 16.7 s. There is a 3-bit additional counter used for the activity timeout, which is 2 min.

The scan key is usually closed, so it draws current normally. It is biased through the 5-V line that is turned off during shutdown, and its keyboard interrupt doesn't need to be enabled for wake up. The Save and Scores keys are biased to battery because they need to be monitored for wake up detection. Either can cause wake up, but polling must be done to ensure that both are pressed. This can be done without activating the 5-V line.

In addition, this can help ignore false key presses. Key press states are determined when two successive key state reads are the same in the Timer0 ISR. Because you need to determine how long a key is pressed as well as if

Listing 1—*Take a look at how the ISR functions.*

```
Timer0_ISR:
;ISR must clear key timers & errorflags when shutdown bit
  is cleared
disable all interrupts (assumes individual int enables
  set elsewhere)
        push A, R0, etc
        clear timer0 interrupt
        increment all active timers
        set timer overflow flags
        read keystates
        set key flags
        check serial port, enable int, etc if enabled
        setup serial port:
ISR detects that serial cable is attached for 2 tmr0 ints
        if so set bit SerialDetect
        then check SerialEn bit
        if enabled then setup port and set bit
  SerialActive
        if not enabled then turn off serial port if al-
  ready active
        check battery:
if not LowBatterySignal then
clear LowBattTimer
clear LowBattery flag
break
else inc LowBattTimer
if LowBattTimer > 5 sec then set LowBattery flag
        check stack overflow:
        if StackVerify <> 7Eh then set StackError flag
        feed watchdog
        pop A,R0, etc
        enable all interrupts
        return from int
```

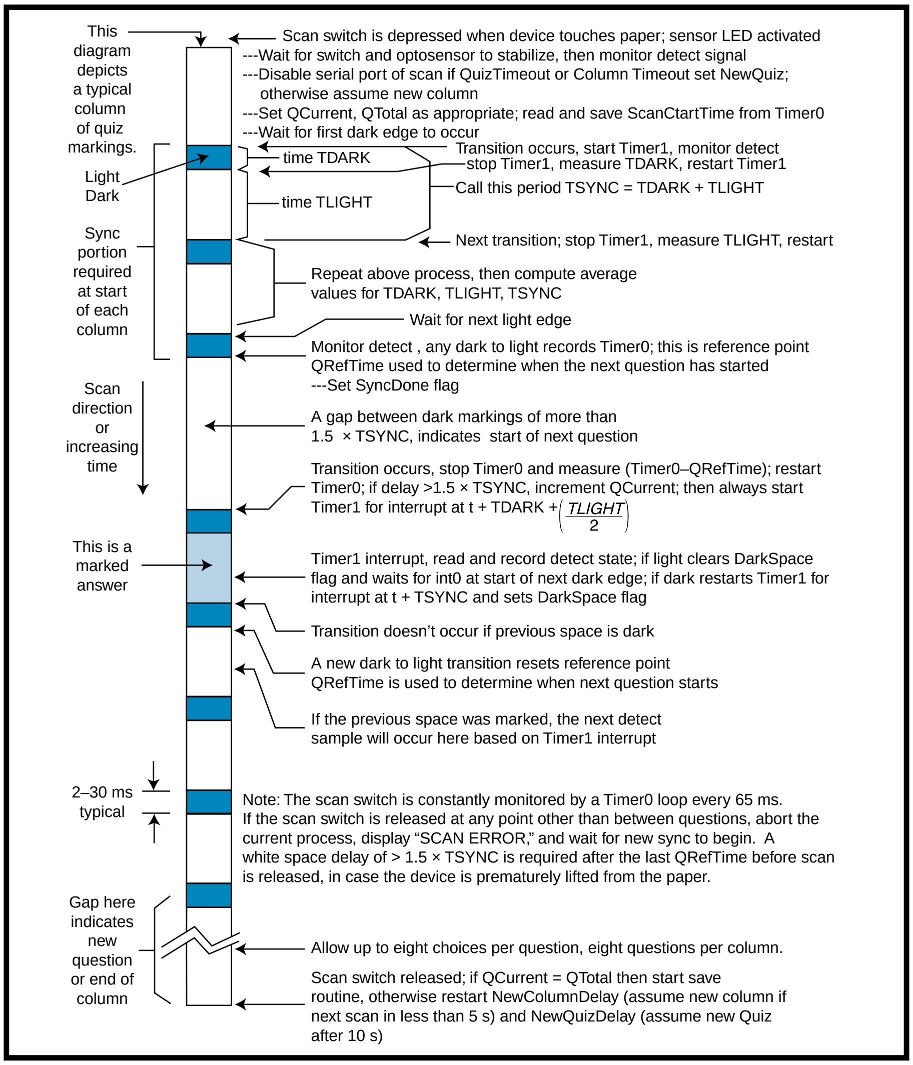

This diagram depicts a typical column of quiz markings.
Scan switch is depressed when device touches paper; sensor LED activated
---Wait for switch and optosensor to stabilize, then monitor detect signal
---Disable serial port of scan if QuizTimeout or Column Timeout set NewQuiz; otherwise assume new column
---Set QCurrent, QTotal as appropriate; read and save ScanCtartTime from Timer0
---Wait for first dark edge to occur
Light
Dark
time TDARK
Transition occurs, start Timer1, monitor detect
stop Timer1, measure TDARK, restart Timer1
Call this period TSYNC = TDARK + TLIGHT
time TLIGHT
Next transition; stop Timer1, measure TLIGHT, restart
Sync portion required at start of each column
Repeat above process, then compute average values for TDARK, TLIGHT, TSYNC
Wait for next light edge
Monitor detect , any dark to light records Timer0; this is reference point QRefTime used to determine when the next question has started
---Set SyncDone flag
Scan direction or increasing time
A gap between dark markings of more than 1.5 × TSYNC, indicates start of next question
Transition occurs, stop Timer0 and measure (Timer0–QRefTime); restart Timer0; if delay >1.5 × TSYNC, increment QCurrent; then always start Timer1 for interrupt at t + TDARK + $\left(\frac{TLIGHT}{2}\right)$
This is a marked answer
Timer1 interrupt, read and record detect state; if light clears DarkSpace flag and waits for int0 at start of next dark edge; if dark restarts Timer1 for interrupt at t + TSYNC and sets DarkSpace flag
Transition doesn't occur if previous space is dark
A new dark to light transition resets reference point QRefTime is used to determine when next question starts
If the previous space was marked, the next detect sample will occur here based on Timer1 interrupt
2–30 ms typical
Note: The scan switch is constantly monitored by a Timer0 loop every 65 ms. If the scan switch is released at any point other than between questions, abort the current process, display "SCAN ERROR," and wait for new sync to begin. A white space delay of > 1.5 × TSYNC is required after the last QRefTime before scan is released, in case the device is prematurely lifted from the paper.
Gap here indicates new question or end of column
Allow up to eight choices per question, eight questions per column.
Scan switch released; if QCurrent = QTotal then start save routine, otherwise restart NewColumnDelay (assume new column if next scan in less than 5 s) and NewQuizDelay (assume new Quiz after 10 s)

Figure 3—*The scanning process diagram shows how the quiz markings are interpreted and how timing is used to detect a marked answer. The sync portion provides an estimate of the scanning speed.*

more than one is pressed, no action is taken on key presses until after they are released.

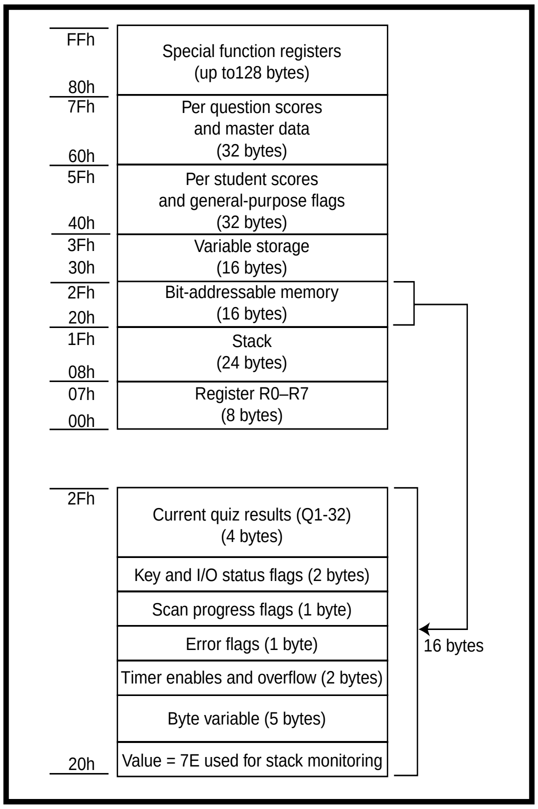
The buzzer and sensor LED can be driven directly from the port pins (20 mA maximum), however, the circuit uses a 2N3904 NPN transistor buffer to provide noise isolation.

The battery voltage was selected to be 4.5 V (i.e., three AAA cells in series) because the LCD's high state output voltage must be within 0.5 V of the processor's supply voltage. Using three AAA cells instead of two is allowed because of the height of the LCD and because the 5-V converter has better efficiency there. A standard AAA alkaline battery (0.41″ × 1.75″) is typically rated for approximately 1000 mAh, or 40 h at 25 mA of continuous usage.

By contrast, you could eliminate the converter by using a standard 9-V battery and a linear regulator. However, that would be less efficient. Furthermore, with a capacity of approximately 500 mAh, it would yield only a 20-h lifetime.

By driving the LCD in 4-bit mode instead of 8, you free an additional four I/O lines, which is convenient and allows you to add power shutdown, battery monitor, buzzer drive, and serial port detection features.

The scan switch could be conveniently replaced by another optosensor, but it would draw too much current. Another alternative is to periodically pulse the scan optosensor (i.e., turn on the LED) and check for a high detect level, indicating that it was sensing a reflective surface. Then, if synchronization wasn't found within a prescribed time, you would turn it off again. However, this also would draw



FFh
80h
7Fh
60h
5Fh
40h
3Fh
30h
2Fh
20h
1Fh
08h
07h
00h
Special function registers (up to128 bytes)
Per question scores and master data (32 bytes)
Per student scores and general-purpose flags (32 bytes)
Variable storage (16 bytes)
Bit-addressable memory (16 bytes)
Stack (24 bytes)
Register R0–R7 (8 bytes)
2Fh
Current quiz results (Q1-32) (4 bytes)
Key and I/O status flags (2 bytes)
Scan progress flags (1 byte)
Error flags (1 byte)
Timer enables and overflow (2 bytes)
Byte variable (5 bytes)
Value = 7E used for stack monitoring
20h
16 bytes

Figure 4—*The internal RAM memory map shows that all of the 87LPC764 memory is used.*

too much current because the sampling would have to be done even if there was no intention of using the device.

The best solution is to embed the scan switch into the tip of the device so that it gets pressed automatically during use. Note that a small lever-activated micro-switch is well suited for this purpose.

Optosensor testing revealed that for the best rail-to-rail output signal, a simple op-amp comparator is needed. The processor's fixed internal voltage reference of 1.28 V is not well centered for this application (2.5 V is desired), and using an external reference would use another I/O pin, so an external op-amp is used instead of one of the processor's comparators. An external op-amp also allows buffering closer to the sensor and away from digital devices for noise protection.

The external op-amp chosen is MC33171, designed for single-supply operation, low 180-µA supply drain, and 1.8-MHz bandwidth. In addition, pin INT0/P1.3 is used for the detect signal, so scanning can be interrupt-driven. This pin is a Schmitt trigger input, which improves noise rejection.

The detect sensor signal must be such that moving the sensor from a light to dark marking causes the detect line to go from high to low, so Interrupt0 can be used (set for edge triggering). When the device is lifted from the paper, there is no reflection and the detect line is low, similar to that of a dark marking. This is not a problem, because the scan switch is depressed first, followed by a short wait before the sensor is read. At this point, detect should be high because of the paper's light area reflections.

SOFTWARE

Results for the current quiz scan are maintained in four bytes, where each bit indicates right or wrong for a particular question (see Figure 4). The cumulative scoring for up to 32 questions as well as the master results require 32 bytes.

An external hardware reset signal is not needed because the QuizWiz is using the watchdog. Whenever a reset occurs, the WatchdogOverflow bit is checked to determine if the watchdog caused it. Because the watchdog timer can't be turned off by the software when it enters shutdown mode, the device must expend a small amount of power to refresh it. Alternatively, the WD timer can be disabled completely at startup and no monitoring is needed during shutdown.

The main ISR loop uses Timer0 in the 16-bit mode 1 and always runs (see Listing 1). The stack will be used for three levels of subroutines. Interrupts will be disabled as needed to ensure this. Assuming that a single subroutine will need to push PC, DPTR, ACC, B, R0, and R1 (worst case), this will require 8 bytes. Therefore the 24 bytes from 08h to 1Fh are reserved for the stack. Store 07Eh at location 20h at the bottom of the bit-addressable memory to monitor stack overflow. This value will be checked during each Timer0 interrupt, and if changed, it will cause a software reset.

LAST THOUGHTS

The most challenging aspect may have been cramming all of the circuitry into the smallest possible package. Several iterations were required to optimize mechanical constraints with battery size, circuit layout, and assembly methods.

The software was limited mainly by the amount of available RAM, but the 8051 architecture proved adequate because of bit-flag memory. If even more RAM was available, then additional scoring details could be stored or perhaps more complex grading results. And, if more I/O pins were available, then you might consider using a full 8-bit LCD interface and a larger character display.

Another improvement would be if the LCD could run directly off battery voltage, which of course varies over time. This might lower current drain and eliminate the need for a 5-V regulator. However, a brief search for such a display was not successful.

The scan timing currently is determined by the use of simple sync marks at the top of each column and, therefore, depends on a uniform scan speed. In a more elaborate approach, the device would determine the scan rate from the timing of the markings themselves and adapt to any changes as the scan progresses.

But, the difficulty is that the processor must store a significant amount of scan data and post-process it to evaluate the variable scan rate and take into account marked spaces.

Paul Kiedrowski obtained a BSEE from Rose-Hulman Institute in 1982 and has worked in the RF, microwave, and communications industries since then. Currently, he is senior staff engineer at Motorola in Fort Worth, Texas, developing base station infrastructure equipment. You can reach him at kiedro@swbell.net.

SOURCE

87LPC764
Philips Electronics N.V.
+31 40 27 34184
Fax: +31 40 27 32250
www.philips.com

NOUVEAU *PC*

Edited by Harv Weiner

SINGLE-BOARD COMPUTER

The Arcom **SBC-386** is a single-board computer designed for use in high-volume applications where networking capabilities are required. It provides the performance and reliability of previous generation boards, with the addition of onboard Ethernet. Applications include communications products, intelligent transportation products, medical products, and industrial control.

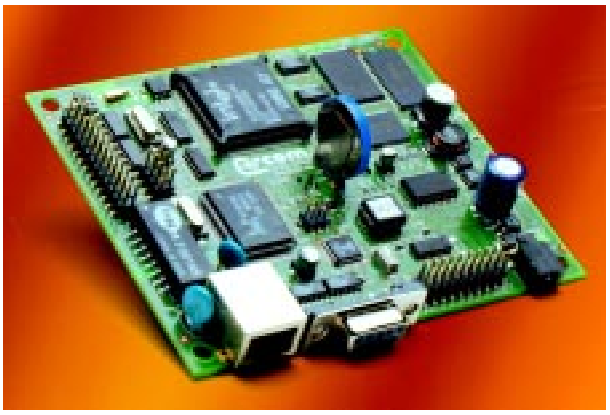
The heart of the computer is the 25-MHz 386EX CPU from Intel. Included on the board are 2 MB of DRAM, 1 MB of flash memory, up to 512 KB of battery-backed SRAM, three serial ports, a debug port, 10BaseT Ethernet, and an onboard 8- to 18-VDC (10- to 16-VAC) power supply.

The SBC-386 is rated at an operating temperature of –20° to 170°C and has been designed to meet European CF standards. The board is supplied with U.S. Software's Supertask Multitasking RTOS and Treck's real-time TCP/lP stack. Arcom also offers a full suite of software development tools.

A development kit that contains all the hardware and software needed to allow immediate development is available.

The SBC-386 (including software) costs **$199** in OEM quantities. The SBC-386 development kit costs **$415**.

Arcom Control Systems
(888) 941-2224
Fax: (816) 941-7807
www.arcomcontrols.com



NOUVEAU*PC*

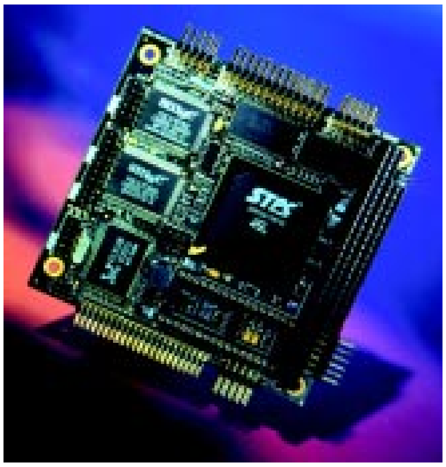
PC/104 CPU MODULE

The **CPU-1220** is an embedded 486 AT computer in a PC/104 form factor.

The module is built around a 486DX core operating at 75 or 100 MHz. Onboard features include 32 MB of DRAM and SVGA with 2 or 4 MB of DRAM (for resolutions up to 1024 × 768 with 16 million colors at 75 Hz). Also included are IDE and floppy disk interfaces, four RS-232 serial ports (two configurable as RS-485), and one 10/100-Mb Ethernet controller.

A keyboard port, bidirectional parallel port EPP-ECP, SSD socket with up to 288 MB of solid state disk, watchdog timer, and real-time clock are included. The parallel port is internally multiplexed with the floppy disk controller, allowing the floppy to be connected.

The flash BIOS has a capacity of 1 MB, 128 KB of which is used for the BIOS and its extensions. The remaining 896 KB can be used as read-only and can store the operating system and user programs and data. The BIOS is reprogrammable onboard, and the setup parameters are saved in flash memory, allowing the module to operate without battery.



The CPU-1220 supports any operating system available for the standard PC platforms such as DOS, ROM_DOS, Windows 3.11/95/98/2000/NT, Linux, as well as real-time operating systems like QNX, pSOS, PHARLAP, VxWork, WinCE, and RT_Linux.

Options for the board include a custom BIOS, fast BIOS boot, extended operating temperature range, and custom connectors.

Eurotech SpA
+39-0433-486258
Fax: +39-0433-486263
www.eurotech.it

EPC
REAL-TIME PCs

Ingo Cyliax

A Cup of Java

Part 1: Embedded and Real-Time Applications

Whether you're starting your morning at Starbucks or capping off the evening at Borders, a cup of joe can make a lot of things better. Don't know beans about Java? Then try Ingo's fine blend of theory and application on the subject.

I might be inclined to think that you've been living under a rock if you've never heard about Java. The Java strategy was developed by Sun Microsystems several years ago. I use the term "strategy" because Java is actually a collection of things (language, object libraries, and Java Virtual Machine), rather than one component.

At first, it seemed like there was a lot of hype about Java, and it wasn't really clear what it was suited for. Initially, it was marketed as a solution to everything. On the client side, Java quickly caught on, with a model to download a Java application (applet) to your web browser and execute it within a safe sandbox kind of environment. This was nice because it allowed you to develop complex GUIs that could run locally and provide solid performance instead of using the traditional HTML-style GUI interface. Also, because the code executes in a sandbox and (by default) only allows Internet connections to the server that the applet is downloaded from, it's easy to develop Internet and Intranet applications without bothering with Internet security issues.

After client-side Java applications became popular, some server-based Java applications gained acceptance, where Java programs can run on a web server to implement applications. Usually this would be a kind of application gateway to a database server, or perhaps a reservation system. There has been a lot of press about these types of applications, but I'm not going to talk about that in this article. What I do want to write about is Java in embedded and real-time applications.

WHY JAVA?

There are several features that make the Java strategy attractive for embedded applications. Java is an object-oriented language that maintains much of the basic C syntax. Think of Java as a C++ with all of the complexity removed. This makes it an easy language to learn and, in theory, you can expend more brain power solving the problems and implementing the algorithms than fighting the syntax of the language.

Another feature is that it is strongly typed, and this is enforced by the compiler and some JVMs or class checkers. I'll get to this a little later.

Besides the strong typing, there is no traditional pointer type. So, like it or not, you can't typecast a pointer to an array into an integer. This is deemed a safety feature because now you can't just access memory willy-nilly, like in C or C++. If you want to access device registers, you'll have to write native functions of an object class. This restricts access to the pointer by using a protocol or API.

Machine type	Length	Java wrapper
—	—	Boolean
Byte	8 bit	Byte
Short	16 bit	Short
Int	32 bit	Integer
Long	64 bit	Long
Float	32 bit	IEEE float
Double	64 bit	IEEE double
Char	16 bit	Unicode character
—	—	Void
—	—	String

Table 1—*Here's a list of the basic data types in Java. These sizes are defined in order to make Java code portable across architectures.*

Finally, the language uses garbage collection instead of relying on the programmer to explicitly free memory. This could be a foreign concept to C and C++ programmers at first, but in practice it's actually not a bad idea.

In a nutshell, you allocate memory when instantiating new objects or arrays. As long as your program is still referencing the object or array, the memory allocated to it stays around. After the reference has been forgotten (i.e., none of the object reference variables or array variables exist), the memory is freed at some point.

Right about now, C and C++ programmers are probably saying, "Yeah, but...." Garbage collection does have issues that need to be addressed when you want to use it in memory-constrained or real-time applications, but it's a slick way of dealing with memory management issues. It pretty much guarantees that there is no memory leak. Memory leaks, along with illegal pointer problems, probably account for most bugs in C and C++ programs.

I will get to these issues next month. This time around, I want to begin with a discussion about the Java programming language and the Java Virtual Machine (JVM).

THE LANGUAGE TO LEARN

As I mentioned, the Java language is fairly simple. There are basic intrinsic data types (see Table 1).

In order to avoid confusion, all of the basic machine data types have defined lengths and encoding. This implies that a long machine type is always a 64-bit integer, no matter what type of machine the code is actually running on. Also, all of the integer quantities are signed integers. There is no unsigned integer.

You should be familiar with all of these data types if you program in C and C++, except for the byte and character type. A byte is an 8-bit integer, and a character is a 16-bit quantity that uses Unicode (rather than ASCII) to encode character representations. The difference is that you can do math on a byte but not on a character. In C and C++ it's possible to do math on a char, even if it holds an ASCII character representation. This is necessary to force programmers to be disciplined about what they mean when they represent a value in their programs.

Listing 1—*This shows an example of a common mistake you can make in C that would not pass the Java compiler.*

```
...
                   int a,b;
               ...
                   a = 1;
               ...
                   if ( a = 1 ) {
                            a = b;
                   }
               ...
```

I've added the types for Boolean, Void, and String, which are not really data types represented in the hardware. Strings are a special kind of type that are implemented in an external class library, so they are objects as far as the JVM is concerned. The compiler knows about them because it needs to be able to deal with String constants. A Boolean is a true/false value, and it's up to the JVM to figure out how to store it in a convenient format.

In Java there is an assignment operator (=) just like in C. In fact, most of the actual language syntax is the same. However, it's stricter about type conversions and whether or not variables have been initialized. For example, the code segment in Listing 1 wouldn't make it past the compiler in Java, but it will pass most C compilers without warning and will actually run.

I don't know how many times I've made this mistake. Granted, a commonly used tool like lint would catch the uninitialized variable in C and C++, but because the assignment in the `if` expression is legal C syntax, it's hard to spot. The reason Java flags it is because the `if` expression requires a Boolean value, but the constant 1 is compatible only with the integer types.

If you're a C programmer, objects may be a new thing for you. Think of them as instances of data types. The object class is similar to a structure definition. When you instantiate an object of a class, the system will allocate memory for the data that is defined in the class. The data elements of a class are called fields. There are a few differences between structures and classes, of course, but these are gravy.

One difference is that, as part of the data type, a class lets you define specific functions you can use to operate on the fields that are stored in an object. These functions are called methods in Java. So, a class is the definition of an object type, which is made up of fields and methods.

Another twist is that you can make fields and methods invisible outside of the class. This way, if you instantiate the object, the implementation details don't have to be visible to the application. All the application has to know is

Listing 2—*Here is a simple* `hello world` *in Java. The source code is more complex than in C, but the object file is only 436 bytes.*

```
import  java.applet.Applet;


public class Hello extends Applet{
  public static void main(String argv[]) {
      System.out.println("Hello World");
  }
}
```

Listing 3—*This example shows that methods in Java can take different argument types and still sort it out. The implementation of* `print()` *has several versions for the different types, and the dynamic linker figures out which one is called based on the signature.*

```
import  java.applet.Applet;

public  class  Count3  extends  Applet{
public  static  void  main(String  argv[])  {
int  i;
System.out.print("Counting...");
for(i=1;i  !=  4;i++)
System.out.print(i);
System.out.println("");
        }
}
```

that there is an object and each instance is a reference to the objects that have been allocated in memory.

Furthermore, you can inherit fields and methods from another class. This lets you build complex classes from simpler classes. Also in Java, a collection of classes that belong together can be bundled into a package. Two examples of packages are `java.lang`, basic classes made up of expected Strings, and `java.io`, classes that are used to perform I/O on the system.

Let's look at some examples of Java and how this might work. In Java, every software module is a class. Even if you want to code a simple `hello world` application (see Listing 2), you have to define a class. In this case, you have to define `hello`.

The `hello` class is based on the `java.applet` class, which gives you the ability to write top-level application classes. In this case, the `hello` class has only one method, main. This method expects an array of Strings, which is the argument list from a command line, and returns void. Notice that in order to use Strings, you do not need to import a class because the compiler knows that Strings are defined in `java.lang.String`. The only thing the application has to do is call an output method to display the `hello world` String.

There's something a little different in Listing 3. Notice that in this example, I print integers as well as Strings using the same method. The methods in a class are matched up using signatures. The signature of a method includes the return value, the name of the method, and the arguments expected. This means that there are at least two different print methods defined in the class `system.out`, public void print(String) and public void print(Integer).

LIGHTS ON, LIGHTS OFF

I built a class and named it `Light`, (see Listing 4). The `Light` class has only one field, the state of the light, which I represent internally as simply on or off. The actual field is hidden from the outside and only the methods, `TurnOn`, `TurnOff`, and `GetStat`, are visible.

If you want to use this class in an application, define the object to create storage (see Listing 5). In this case, each object contains the state field, so it's small. To use it, call the methods that have been exported by the class.

Now you have a class that behaves as if there's a light somewhere. In fact, I defined several lights in this example. This is great for testing your system before you have the hardware to turn lights on and off. When the hardware becomes available, you can replace the simulated `Light` class with one that implements talking to hardware. Listing 6 shows that I changed the code to turn the lights on and off by making a call to poke, which is a native call that toggles bits in a hardware port.

This illustrates one use for object-oriented programming. If you combine the hardware components or subsystems into objects, it's easy to replace objects that implement a test version of the real thing. In Java, it binds classes at run time, rather than at compile time. So replacing the test function would simply involve copying the right class file into place, or changing the search path to the class file. Whenever you start the software, it will automatically find the appropriate class and run.

If you're seriously considering learning Java and have some C or C++ experience, I recommend reading *Thinking in Java*, by Bruce Eckel, which is also available online. [1]

CLASS FILE

After you've written a module in Java, you need to compile it into a class file. This is the object file format for Java and normally ends in `.class` extensions. A class file contains all of the information a module needs to be integrated into another program or class. It also includes what other classes and modules it depends on and hints to where the JVM can find them.

Listing 4—*This is the main program for the Light example. It uses a class called `Light`, which has three interface methods, `TurnOn`, `TurnOff`, and `GetStat`.*

```
import  java.util.*;
class  Light  {
     int state;
     void TurnOn() {
         state = 1;
    }
     void TurnOff() {
         state = 0;
    }
     String State() {
         if (state == 0)
            return "Off";
        else
             return "On";
    }
}
```

In the examples I've provided, the class file that contains the main method is the one you have to start with. To run these programs, you need to tell the JVM to execute this module. Typically, Java hello will do the job. The JVM will load the class file and look for the main module that matches the signature Void main(String []). If there is no such method in the top-level class, it will not start. When found, it will start executing the code for the main method. When the JVM finds a reference to a method or field, it will look in the class files symbol table to figure out where to find the class file to resolve that reference. The JVM then loads that class file and executes the referenced method or accesses the field. If an object of that class is instantiated, then the class will get loaded when the JVM allocates the memory for it, because it needs to resolve how much memory the object requires.

A CLOSER LOOK

Up to now I've mentioned the Java Virtual Machine in passing, so you're probably wondering what it really is. All Java class files contain machine instructions for a fictional processor (virtual machine). Tim Lindhol and Frank Yellin of Sun Microsystems have written a document called "The Java

Listing 5—*This is the software-only implementation of the `Light` class. It uses a state variable to track the state of a virtual light.*

```
import  java.util.*;
import  java.applet.Applet;
import  Light.*;
public  class  House  extends  Applet{
public  static  void  main(String  argv[])  {
   Light  bedroom  =  new  Light();
    bedroom.TurnOn();
System.out.println("The  Bedroom  light  is
"+bedroom.State()+"."  );
    bedroom.TurnOff();
    System.out.println("The  Bedroom  light  is
"+bedroom.State()+".");
  }
}
```

Listing 6—*The hardware version of the* `Light` *class uses native C function calls to actually twiddle some bits on a hardware port. It has the same name and interface as the software-only version, and the application picks up whichever one is the default in the current directory.*

```
import java.util.*;
   class Light {
     void TurnOn() {
       outb(0x378,inb(0x378)|0x01);
    }
  void TurnOff() {
  outb(0x378,inb(0x378)&~0x01);
 }
  String State() {
   if ((inb(0x378)&0x01) == 0)
   return "Off";
  else
   return "On";
 }
  static native int inb(int a);
  static native void outb(int a,int v);
}
```

Virtual Machine Specification," which describes the class file format and what each instruction needs to do in order to be compatible. [2]

The JVM is stack-based and has several different classes of instructions that implement the basic data types (byte, char, int, short, long, float, and double). There are mathematical instructions (add, subtract, etc.), logical expressions (and, or, xor), conditional jumps, support for arrays, and instructions to deal with field accesses and invoking methods. The last class of instructions is the most difficult to implement because fields and methods are specified symbolically, and the instructions that deal with them have to contend with the dynamic loading and linking of the classes.

NATIVE MACHINE CODE

Many compilers produce intermediate code that can be interpreted or converted into native machine code for an architecture. There are concerns when you interpret intermediate code, as is true with naive JVMs. However, most Java instructions can be translated into native machine instructions efficiently. This code translation can be done in several ways.

One way is to statically convert all of the code in a class into native machine code. This is typically done with code that you don't expect to change for a particular architecture. The drawback is that Java byte codes are dense and memory efficient. After translated into machine code, the code takes up more space, which might be a problem if your device is memory-constrained.

Another way is to compile the code dynamically, because then the Java methods are translated by the JVM into native machine code before it executes them. This can be done when the class is first loaded, or even when a method is executed the first time. The latter is sometimes referred to as just-in-time (JIT) compilation.

JIT COMPILATION

There are problems with JIT compilation as well. Obviously, there is a slowdown the first time a method has to be translated before being executed. Some JIT-based JVMs try to analyze how often a method is being called before compiling it into native machine code. The strategy is that there is some threshold when translating the method, and running it in native is cheaper than interpreting it. Some JITs will optimize the code after it's translated. When translated, you have to allocate memory for the native code.

Performance-enhancing techniques were developed for desktop and server applications where you need the best performance. However, for embedded and real-time applications, JIT and static compiling methods will get in the way. The natively translated code is likely to be bigger than the Java byte codes. All of the Java instructions are expressed in instructions that are 8 bits in length. Also, because the machine is stack-based, many frequently-executed instructions do not have operands. This makes Java byte codes dense.

Also, doing the dynamic compilations or JIT introduces uncertainty about timing. The first time a method is invoked or a class is loaded, the system will go away and translate the code. Some JITs will cache translated code blocks and throw out code if it's not refreshed by executing often enough. Clearly, this won't work for real-time applications.

For those, you typically want to statically pre-compile your methods to native code or just use a JVM that does not perform JIT compilation on methods that need to run in real time.

Fortunately, you don't have to write your own JVM implementation, unless you're planning to run Java on a new architecture that isn't supported. There are many commercial and noncommercial JVMs. Sun implemented JVMs on several architectures. IBM and Agilent have JVMs. On the real-time or embedded front, Newmonics and Wind River have Java-based solutions. There are JVMs for 8- and 16-bit microprocessor platforms like Dallas Semiconductors' Tini or Smart Network Devices. Finally, vendors like Ajile, Sun's PicoJava, and Derivation Systems sell hardware Java solutions that execute Java byte codes directly.

ON THE HORIZON

Next month, I plan to write about some of the real-time issues in Java. In the third article, I will look at some of these JVM solutions in more detail.

As usual, when I introduce a new topic, keep in mind that a single tool won't solve every problem. Just as there are various CPUs and operating systems, there are various languages.

With this said, think of Java as yet another tool you can put in your embedded systems engineering toolbox to help you solve problems. Java makes it easier to code large projects in a short time, but remember that there are some performance issues. ◾

Ingo Cyliax is a computer and electrical engineer (BSCEE) and the founder of EZComm Consulting, which specializes in embedded systems and FPGA design services as well as troubleshooting. Ingo has been writing about various topics ranging from real-time operating systems to nuts-and-bolts hardware issues for several years. He can be reached at cyliax@ ezcomm.com.

REFERENCES

[1] B. Eckel, *Thinking in Java*, Prentice Hall, Upper Saddle River, NJ, June 2000.

[2] T. Lindhol and F. Yellin, *The Java Virtual Machine Specification*, Addison-Wesley, Reading, MA, 1997.

EPC
Applied PCs

Fred Eady

Rabbit Season

Part 4: The Wonderful World of TCP/IP

With only one more part to go in this series, Fred wants to cover a few more of the Rabbit Ethernet project details. So, this month he shows us how the board handles TCP/IP and everything from collisions to pointers to calling home.

Frank Sinatra did it his way, and last month, so did I. In my previous article, I took a CS8900A-CQ Ethernet IC and strapped it to the back of a Rabbit. The Ethernet IC and silicon bunny combination worked well, although the intercard wiring between the Rabbit and my Ethernet development board was a bit tricky. As I pointed out earlier in this series, there's more than one way to force a Rabbit to generate Ethernet packets. Move over Frank, there's one more Rabbit coming to the stage in this magic show. Tell Alice to bring along her teacup in Wonderland, and let's follow that silicon hare (wearing a TCP/IP stack over its ears) as it hops down the Ethernet trail.

DYNAMIC C PREMIER

The Rabbit 2000 TCP/IP development kit is a single-board computer that is based on the Rabbit 2000 microprocessor. Although it comes out of the box with its own version of Dynamic C, I happened to have a version of Dynamic C Premier that I used instead. Dynamic C Premier is contained on a single CD and is accompanied by 1.2″ of excellent documentation. The manual set consists of an introduction to TCP/IP, TCP/IP high-level protocols, TCP/IP function reference, and the Dynamic C Premier user's manual.

Everything for the Rabbit 2000 hardware set is included with Dynamic C Premier. All of the sample code and libraries for the standard Rabbit 2000 Basic development kit, the Rabbit 2000 TCP/IP development kit, and the Rabbit 2000 core module are shrink-wrapped into the package.

Dynamic C Premier also contains full TCP/IP source code. Almost all of TCP/IP's rowdy friends are in on this party. ICMP and HTTP with facilities for SSI, CGI routines, cookies, and basic authentication are all rolled in. Also joining this magic show are SMTP, FTP, and TFTP (client and server) capabilities. The Rabbit TCP/IP SBC wouldn't rate without an Ethernet interface, so, just to keep me happy, the folks at Rabbit Semiconductor dropped Ethernet drivers for the Realtek NE2000 chipset into the Dynamic C Premier package.

As you read on, you'll see that I'm going to use some of this TCP/IP and Ethernet driver software magic to

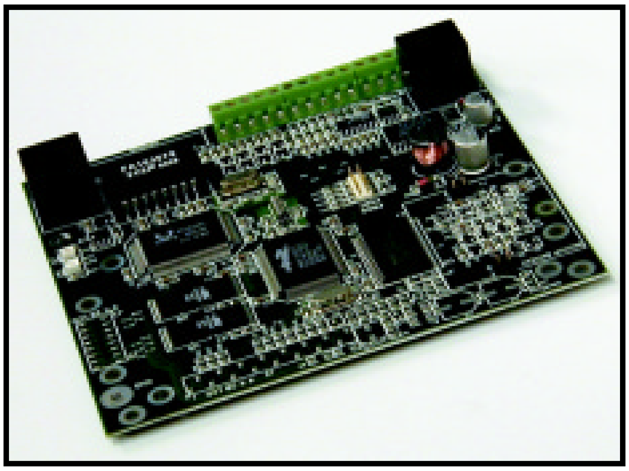


Photo 1—*This proves it. They really aren't invisible when they sit still!*

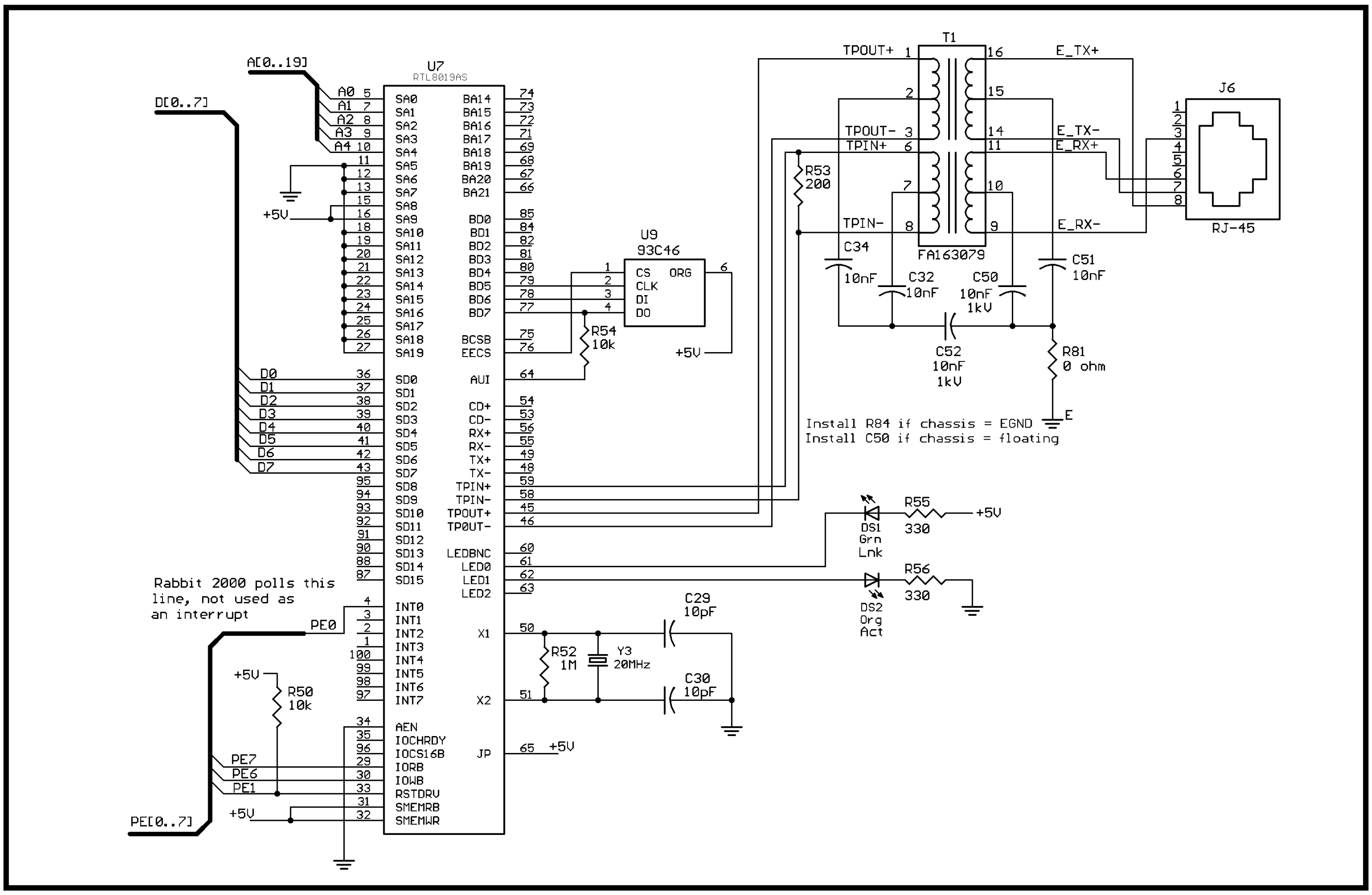


Figure 1—*An extra address line for the RealTek 8019AS's internal register set is the major physical difference between implementing the RealTek 8019AS and the CS8900A-CQ.*

implement a hare-raising Internet appliance that monitors its environment and sends e-mails to inform you of the good, the bad, and the ugly.

The microprocessor bunny on the Rabbit 2000 TCP/IP Development Kit hops along at 18.432 MHz. A 10BaseT Ethernet interface is about the only thing that logically differentiates the Rabbit 2000 TCP/IP Development Kit from the Rabbit 2000 Basic Development Kit. All of the timers, I/O ports, and memory resources that come with the latter are included in the former as well. A still TCP/IP Rabbit is captured in Photo 1.

REALTEK 8019AS

Unlike your CS8900A-CQ-based home-brewed Ethernet bunny, the Rabbit 2000 TCP/IP Development Kit uses the Realtek 8019AS. The Realtek 8019AS is an inexpensive NE2000-compatible Ethernet IC that's based on the National DP8390 Network Interface Controller, which (like the CS8900A-CQ) provides all the media access control layer functions required for transmission and reception of packets in accordance with the IEEE 802.3 CSMA/CD standard.

In 8-bit mode, using DMA resources to transfer data from the NIC's on-chip buffer memory to the host microprocessor-controlled memory is not an available option on the CS8900A-CQ, but DMA is used to an advantage on the Realtek 8019AS.

Within the Realtek 8019AS, the onboard FIFO and DMA channels work together to form a simple packet management scheme that provides up to 10-MBps internal DMA transfers.

A second set of Realtek 8019AS DMA channels is provided to get data out of the internal buffer ring and into the microprocessor-controlled memory for processing. The first set of DMA channels is termed local and the second set is called remote. Local DMA channels burst data into and within the Realtek 8019AS. Remote DMA channels help get data from the Realtek 8019AS NIC to the host microprocessor.

Taking a look at Figure 1 and comparing it to the CS8900A-CQ implementation in earlier articles, you can see that physically the Realtek 8019AS and the CS8900A-CQ hook up to external support circuitry in an almost identical manner. Just like the CS8900A-CQ, the Realtek 8019AS was originally designed for major Ethernet applications in desktop personal computers. 10/100 Ethernet ICs came along and washed out the usefulness of the Realtek 8019AS as a major desktop player. Because of its affinity to embedded-oriented 8- and 16-bit microprocessors, the Realtek 8019AS (like the CS8900A-CQ) found its way into embedded applications.

ALMOST ONE AND THE SAME

The Realtek 8019AS is controlled through an array of on-chip registers. It isn't CS8900A-CQ PacketPage technology, but it's logically identical. Again, just like inside the CS8900A-CQ, the Realtek 8019AS registers are used during initialization, packet transmission, and reception. There are also registers for remote DMA operations on the Realtek 8019AS that don't exist on the CS8900A-CQ. By using the Realtek 8019AS internal registers you can perform the same logical operations as when using the CS8900A-CQ's PacketPage registers during initialization. These operations include defining the physical address, setting receive and transmit parameters, and masking interrupts.

On the Realtek 8019AS, add configuring DMA channels and hacking out transmit and receive buffer ring areas to the aforementioned list. Because DMA is an integral part of the Realtek 8019AS's NIC-to-microprocessor interface, there must be a control mechanism or register to act as the traffic cop for the dataflow between NIC and microprocessor memory and the NIC-to-Ethernet interface. This is done within the Realtek 8019AS via the command register (CR), which is used to initiate remote DMA operations as well as data transmission.

On the original CS8900A-CQ design I presented, the interrupt line was polled by the microprocessor to check for receive activity. If you take a closer look at Figure 1, you'll see that the Rabbit 2000 TCP/IP Development Kit follows the CS8900A-CQ logic that polls the NIC interrupt line. Only the names have been changed to protect the innocent. The microprocessor senses the Realtek 8019AS interrupt via polling and consults the Realtek 8019AS's interrupt status register (ISR) to determine what type of interrupt has occurred.

The CS8900A-CQ generates interrupts in the same way, except the register called the ISR on the Realtek 8019AS is called the the Interrupt Status Queue on the CS8900A-CQ. Hmm…I wonder if they call the top on an IC a "bonnet." (For those of you who think us Southern boys aren't "edumacated," a bonnet is the common term for a hood when cars are the topic of discussion.)

COLLISION COURSE

We're working with Ethernet here and dealing with it is similar to watching a NASCAR Busch Grand National Race. It's a foregone conclusion that collisions are going to occur. So, the CS8900A-CQ and Realtek 8019AS transmit packets are in accordance with the CSMA/CD protocol. Both the Realtek 8019AS and the CS8900A-CQ schedule retransmission of packets on collisions up to 15 times according to the truncated binary exponential backoff algorithm. The CS8900A-CQ datasheet calls this the standard backoff algorithm. After you cut the transmit process loose, each NIC is on its own until the transmission cycle is aborted or completed.

Assuming that buffer memory is free for the taking, transmitting packets with the Realtek 8019AS entails setting up an IEEE 802.3 conformal packet in memory with 6 bytes of the destination address, 6 bytes of the source address, the data byte count, and data.

If you came in on the ground floor of this set of articles, you may recognize the above sequence of bits as a standard Ethernet packet. When the required packet items are laid in by the controlling microprocessor, the following Realtek 8019AS registers are loaded with TPSR (the packet starting address) and TBCR0 and TBCR1 (the length of the packet).

Finally, to initiate the transmission, the PTX (transmit packet) bit of the Realtek 8019AS command register is set. If the total length of the Ethernet packet is less than 46 bytes, the Realtek 8019AS (and the CS8900A-CQ) will automatically pad the packet to avoid sending a runt packet onto the network. If you're into research and debugging, by configuring the right bits in the right registers, you can tell either the CS8900A-CQ or the Realtek 8019AS to send the runt anyway.

All of the Realtek 8019AS acronyms I just exposed you to can be effectively put into place by following the setup in Figure 2.

The CS8900A-CQ is a little different in that the on-chip buffer space is asked for and permission is granted to load the buffer area before any data is transferred for transmission. This is called a bid for buffer space in the CS8900A-CQ documentation. The Realtek 8019AS documentation stresses that if the buffer ring area of the Realtek 8019AS is set up correctly at initialization, there should never be any contention for transmit buffer memory during normal operations.

The transmission of data to the ether is an area that both NICs (the CS8900A-CQ and the Realtek 8019AS) run in parallel as far as operation is concerned. Before jumping into the ether, both NICs will check themselves to see if they are receiving.

If the coast is clear, the CS8900A-CQ starts transmission of the 8-byte preamble after a specified number of bytes (5, 381, 1021, or all) are loaded into the transmit buffer. Note that this threshold is determined by bit settings in the CS8900A-CQ Transmit Command Register.

The Realtek 8019AS uses its DMA channels and FIFO to follow the preamble with valid data. The Realtek 8019AS DMA bursts data to the FIFO, which is then serialized out as clocked NRZ data. Then, the Realtek 8019AS DMA refreshes the FIFO when the FIFO additional maximum threshold is reached. The FIFO threshold is register-programmable.

In both cases, each NIC continues the transmission as long as the transmission byte count in the registers are greater than zero. After all bytes are sent, the CRC is calculated by each NIC and sent to complete the packet.

If a collision occurs during transmission, the transmission is stopped and 32 ones (a jam sequence) are transmitted to make sure the segment knows a collision just took place. The standard backoff algorithm is executed by the CS8900A-CQ and the Realtek 8019AS, and the transmission is tried again. When the transmission finishes, both NICs have transmit status registers that can be queried as to how the transmission went.

Receiving data from the ether is a similar process for each NIC, as well. Both NICs listen to the wire, sense a carrier, and start synchronizing with the alternating ones and zeros pre-

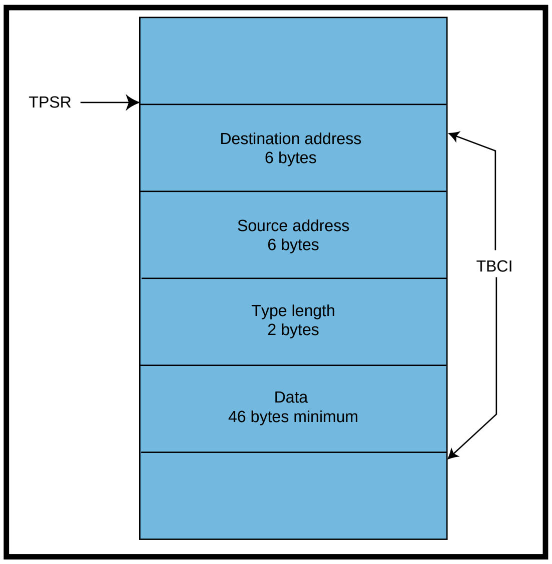


Figure 2—*You can fiddle around with the transmit buffer's location and size on the RealTek 8019AS. This is fixed on the CS8900A-CQ.*

amble. After the two consecutive ones of the SFD (start of frame delimiter) are sensed, the preamble ends and the NICs expect everything behind this set of ones to be valid data. Both NICs check the destination address to see if the incoming packet is addressed to them. If it's not, then the packet is moved into buffer memory and subsequently discarded.

On the other hand, if the packet destination address matches the NIC's address filter setting (hashed or individual), the packet is moved into memory so it can be transferred to the microprocessor memory for processing. If everything goes OK during the receive cycle, both NICs post receive statuses in their respective registers and generate an interrupt to let the microprocessor know that an event needs attention.

SOME POINTERS

The Realtek 8019AS differs from the CS8900A-CQ in that the data coming into the Realtek 8019AS is put into a receive buffer ring, whereas the CS8900A-CQ stuffs the data into a flat predefined buffer area. The CS8900A-CQ method is valid and there is nothing special about the Realtek 8019AS's ring buffer. It's a classic circular, head and tail buffer scheme. The following four pointers control the ring:

- PSTART—the beginning address of the buffer ring
- PSTOP—the address at the end of the buffer ring
- CURR—the current page pointer, which points to the next available buffer area for the next incoming packet
- BNRY—the boundary pointer, which points to the next packet to be unloaded from the buffer ring

The buffer ring size is determined by the bytes between PSTART and PSTOP. PSTART and PSTOP are loaded at initialization time. As packets come in, the CURR pointer moves ahead of the BNRY pointer around the ring. If CURR reaches BNRY, the buffer ring is full. All receptions are aborted and missed packet registers are updated until this condition is cleared. The remote DMA channels help remove packets from the buffer ring. The original datasheets tell you directly that enough receive buffer memory should be allocated initially to avoid the CURR = BNRY condition.

Each Realtek 8019AS buffer is 256 bytes long. A valid received packet and a 4-byte offset are placed at location CURR. Buffers are automatically linked together to receive packets larger than 256 bytes. When all the bits are in, the receive status register (RSR), a pointer to the next packet, and the byte count of the current packet are written into the 4-byte offset. That's basically how the Realtek 8019AS works.

To be honest, I was disappointed with the Realtek 8019AS datasheets and support. I realize that, as a professional, I'm supposed to know about parts like the Realtek 8019AS. So far, I haven't received an airline ticket to Taiwan for a personal briefing. So, how do other people learn how to manipulate this IC? Besides, who doesn't know that NE2000 and DB8390 don't correlate?

The Realtek 8019AS datasheet on the web site leaves much to be desired. It states, "This documents only the differences between the 8019 and the 8019AS." Nothing plus nothing equals nothing. So, you know me, I fired off an e-mail to Realtek asking for more detailed documentation. They pointed me to the DP8390 where I looked at a National AT/Lantic datasheet. This turned on the gas but didn't boil the water. Not yet satisfied at this point, I wrote a second note to Rabbit Semiconductor. I figured if they were using this part, they had to know how to make a go of it.

JUST LIKE MAGIC...

Well, what do you think I ended up with? If you guessed more DP8390 stuff, you're right! In the end, everything I received from all parties proved useful. I found that the code offered in the National DP8390 datasheets from Rabbit Semiconductor was portable as well as informative. All of the datasheet material descriptions are posted at the end of this article for your reading enjoyment.

I'm amused that, by putting the NE2000-compatible label on the Realtek 8019AS, you're expected to know what to do with it. Get a grip. I'm not caught in an episode of Star Trek here but if I was, my TriCorder must not be working. If you want to see how the Rabbit 2000 TCP/IP development kit brings up its onboard Realtek 8019AS, just start some of the Rabbit's TCP/IP sample code and step through it. When you do that, the Realtek 8019AS packet driver source code will magically appear.

RABBIT PHONES HOME

Now that you have a sense of what the Rabbit 2000 TCP/IP development kit is and some knowledge of the Realtek 8019AS Ethernet IC that resides on it, let's get on with a description of the supporting hardware you'll use to make it send e-mail.

If your Rabbit was in a briar patch (a remote location void of other friendly cottontails), and it sensed something was not as it should be, it would most likely hop to the nearest telephone to dial, connect, and log on to the ISP, compose an e-mail, and fire it off. When the e-mail was en route,

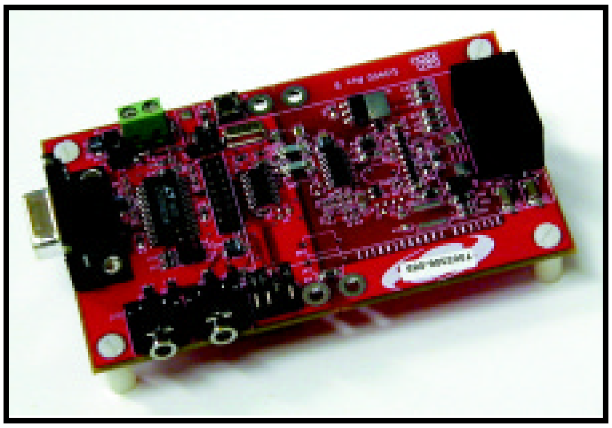


Photo 2—*I've never heard bunnies babble, but if they do, the Si2400 ISOmodem can send it over.*

your silicon bunny would return to its post in the briar patch to listen for events to write home about. The problem is, there's no modem included with the Rabbit 2000 TCP/IP development kit to help any of the onboard serial ports send data or e-mail via dialup through the Internet. And, if there was a modem, there's no dial-up IP software to make the connection to the ISP. Basically, it would be out there collecting intelligence at Rabbit speed with no way of communicating its findings to Cottontail Central.

Well, I've been talking about development platforms from the beginning of this article. So, why not bring on one more development board to solve the modem problem?

You'll need a modem that can do what desktop modems do in an embedded size and price range. For the e-mail packets the bunny will be sending, the modem you select won't have to be fast, but it would be nice to have all the goodies you've come to love on internal and external personal computer models. For instance, the AT command set is a lot easier to deal with than some obscure proprietary modem command set.

Another factor is the location of the briar patch. Your modem should be just as at home with a Texas jackrabbit as it would be with a nose twitching Aussie puddle jumper. Thanks to my peers at *Circuit Cellar*, the search for a suitable modem ended quickly; Jeff Bachiochi wrote about the Si2400 in issue 117. Silicon Labs' V2.BIS ISOmodem is imitating an invisible Rabbit in Photo 2.

ISOMODEM

What you see in Photo 2 is the Si2400 evaluation board. Looking closer at the photo you'll notice two SOIC packages, the Si2400 and the Si3015, separated by a couple of large SMT capacitors. Those four parts are essentially the whole enchilada. Actually, the Si2400 ISOmodem is a chipset that integrates a globally programmable telephone line DAA (direct access arrangement). Normally, I list things that are included with a whiz-bang part. With the Si2400 ISOmodem chipset, the following are what you don't need to include:

•DSP data pump
•modem controller
•AFE (analog front end)
•isolation transformer
•relays
•optoisolators
•2- to 4-wire hybrid
•voice codec

Now here are the things the Si2400 ISOmodem chipset does without the previous list of items:

•2400-bps V.22bis
•1200-bps V.22, V.23, Bell 212A
•300-bps V.21, Bell 103
•V.25 fast connect
•V.23 reversing
•security protocols including SIA
•Caller ID detection and decode
•DTMF tone generation and detection
•UART functionality with flow control
•globally programmable integrated DAA with parallel phone detect, overcurrent protection, and capacitive isolation
•AT command set
•integrated voice codec
•call progress support
•HDLC framing in hardware
•PCM DATA pass-through mode
•3.3- or 5-V power

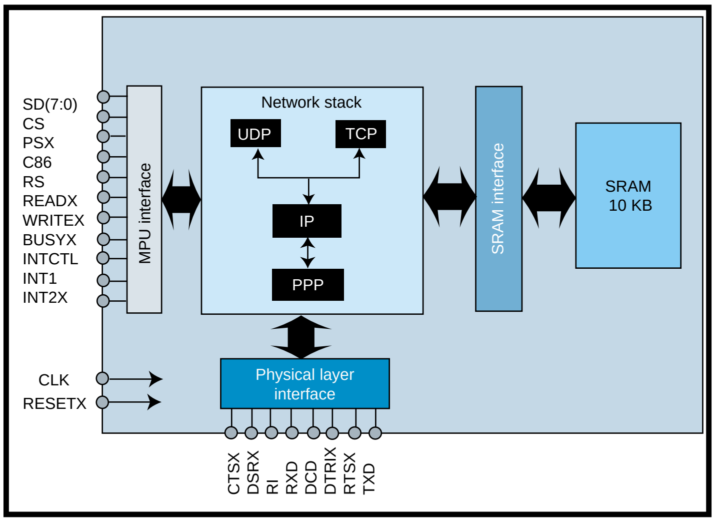


Figure 3—*There's not much I need to say about this picture.*

It's a good thing Rabbits can't speak, or the Si2400 ISOmodem would come as standard equipment on every bunny in the field. Voice codec is nice, but I won't be using it here.

The silicon bunny now has something to say and a new modem, but with the equipment it has now, it can't get any e-mail through to the ISP. That's because there's no protocol in place to complete the ISP connection. If you back up a bit, you'll notice that I didn't mention anything about PPP (point-to-point protocol) being a Rabbit virtue. Most ISPs use PPP for dial-up connections today. Well, I heard on the QT that the Rabbit programming staff is close to releasing PPP code for the Rabbit 2000 TCP/IP development kit. Just in case it doesn't hit the streets by press time, I'll revert to Plan B from the Florida room. Plan B entails the implementation of Seiko Instruments' S7600A engine for the Rabbit (see Photo 3).

The iChip S-7600A in Photo 3 has a TCP/IP protocol stack built into its hardware that provides TCP/IP functionality for small microprocessors that are unable to support a full TCP/IP stack and their application coding at the same time. The Rabbit is not included in the small microprocessor equation because it can indeed support a full TCP/IP stack and carry on with its daily duties as well. Right now, all you need the S-7600A to do is provide the PPP functionality that hasn't been found in the Rabbit's winter coat yet.

PLAN IN MOTION

As you can see in Figure 3, the S-7600A takes Rabbit data in and spits PPP, TCP/IP, and UDP out the other end to a modem interface. I'll have to side with Steve Ciarcia on this one. I'd rather solder in the S-7600A than write the TCP/IP stack code.

So, the plan's set. The Rabbit 2000 TCP/IP development kit includes the capability to send e-mail using SMTP code in the Dynamic C Premier libraries. The Rabbit 2000 TCP/IP development kit has ample I/O to sense events that will trigger and send e-mails. Si2400 ISOmodem hardware is standing by to affect the ISP connection and the S-7600A is ready to fill in if you don't get the PPP native code for the Rabbit 2000 TCP/IP development kit before press time.

The ISOmodem evaluation board is available for $150. Considering the need for FCC approval of the DAA, I opted to purchase the eval board for this development project. I held off on the S-7600A evaluation kit because $199 was too expensive for what it offers in terms of hardware, and it assumes that an ISA interface would be used for evaluation. I wasn't ready to pay for an ISA FPGA I may never use. Seiko did stick a standalone S-7600A on the board but that wasn't enough to sway me. So, next time I'll construct a nifty little S-7600A board we all can afford, send e-mail from the Rabbit 2000 TCP/IP development kit, and again prove without a shadow of a doubt that it really doesn't have to be complicated to be embedded. ■

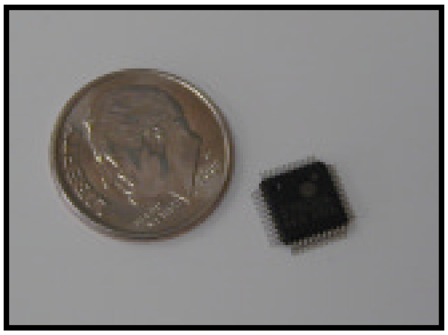


Photo 3—*Here are 48 pins of Internet Express.*

Fred Eady has more than 20 years of experience as a systems engineer. He has worked with computers and communication systems large and small, simple and complex. His forte is embedded-systems design and communications. Fred may be reached at fred@edtp.com.

RESOURCE

Application notes
AN-475, AN-874, AN-93

SOURCES

Realtek 8019AS
Realtek Semiconductor Corp.
(886) 3 5780211
Fax: (886) 3 5776047
www.realtek.com.tw

Si2400 ISOmodem
Silicon Laboratories
(877) 444-3032
Fax: (512) 416-9669
www.silabs.com

S-7600A
Seiko Instruments USA
Electronic Components Division
(310) 517-7771
Fax: (310) 517-7792
www.seiko-instruments-usa.com

Ethernet development board
E D Technical Publications
(800) 499-3387
Fax: (321) 452-1721
www.edtp.com

Rabbit 2000 TCP/IP development kit
Rabbit Semiconductor
(530) 757-8400
Fax: (530) 757-8402
www.rabbitsemiconductor.com

FEATURE ARTICLE

Brian Millier

Using a T6963 Controller-Based Graphics LCD Panel

With the latest advances in display-panel technology, you wouldn't think that Brian would have a hard time finding a simple 200 × 64 pixel display. But, finding one that was easy to design into a micro-based project had its challenges.

Rodney Dangerfield's favorite expression is "I don't get no respect," a sentiment I'm sure LCD panels would share if they had feelings. Consumers own so many gadgets containing LCDs that they don't give them a second thought. Like most electronics enthusiasts, I take them for granted now that I've used so many $10 alphanumeric LCDs. Had I not recently replaced my laptop with one that features a 15″ XGA screen, I probably would feel blasé about that LCD panel, too.

When faced with designing an instrument incorporating a modest-sized graphics LCD, this cavalier attitude evaporates quickly. I recently designed an instrument that needed a graphics display with about 200 horizontal pixels and at least 64 vertical pixels. My first thought was that there must be tons of small LCD panels languishing in warehouses, the victims of a technology moving so quickly that many products become obsolete almost as soon as they are introduced. Unfortunately, many of those LCD panels are hard to design into the average micro-based project.

In this article, I'll outline the experience I gained while designing a "serial backpack" that makes it easy to use a popular LCD panel in small micro-based projects.

THE PLAYERS

I searched the Internet for available devices and learned that they fall into three broad categories including VGA and sub-VGA panels, NTSC video panels, and LCD panels that contain onboard controller and memory.

The first panels, VGA and sub-VGA panels, lack both an "intelligent" controller and raster memory. They are difficult to use in a project containing only a conventional microcontroller because they must be constantly refreshed and the timing is critical. They are designed to use with a dedicated controller LSI chip and usually are interfaced to a full microprocessor bus. I tip my hat to those *Circuit Cellar* authors who have managed to use this type of display with a PIC micro and still have some CPU time left to do useful tasks. Although not for the faint-hearted, their advantage is that they can be obtained on the surplus market for a song.

NTSC video panels are designed to be driven with standard TV video signals and have an aspect ratio similar to that of a standard TV screen. They come in all sizes from 0.75″ to about 5″ and cost about $50 on the surplus market. These displays are best left to applications where NTSC video is the main requirement. Incidentally, if you need video with a text overlay, consider Decade Engineering's BOB-II.

The third category, LCD panels that contain an onboard controller and memory, generally use a CMOS LSI controller such as the Toshiba T6963, Sanyo LC7981, S-MOS SED133x series, or Samsung KS0708. Unlike the devices mentioned earlier, these units have onboard raster memory, which means that they don't require constant refreshing. This relieves your project's micro of a time-consuming chore. Of the three controller chips, the Toshiba T6963 has the best mix of features, availability, and documentation.

THE T6963 CONTROLLER

The T6963 controller isn't an "intelligent" controller, so don't go looking for a screen clear command, line

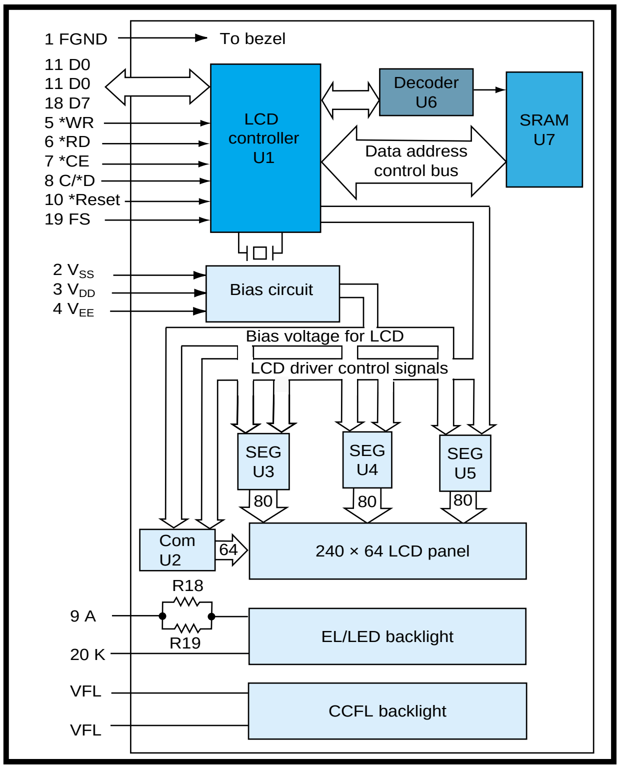


Figure 1—*Although not labeled as such, U1 is a Toshiba T6963 controller in the AZ Displays' AGM2464D LCD panel.*

draw command, or automatic cursor handling. Anything more involved than putting up a text character on the screen requires firmware routines. The good news is that the T6963 has a well-designed memory architecture and instruction set that provide efficient text and graphics routines. Unlike some other controllers, the T6963 has a character generator built-in.

An important feature is the screen memory organization. The T6963 can control a 64-KB SRAM directly. Commonly this controller is used with 240 × 64-pixel LCD displays and uses an 8-KB SRAM, which provides at least twice as much memory as is needed to perform all functions.

Another significant feature of this controller is that it uses separate memory areas for its text and graphics bitplanes. You can, for example, erase text without disturbing underlying graphics images. Or, you may draw dynamic graphics displays such as charts and graphs without worrying about clobbering text labels in the same region of the screen.

The custom character generator area (CGRAM) is a third memory area. In normal text operation, the T6963 uses an internal ROM-based character generator for the first 128 characters of its font. The last 128 characters are defined by patterns that you may download into the CGRAM memory.

There also is an external character generator mode for users who want to define a complete 256-character font. This could be useful for a foreign language, for example. Eight bytes per characher of SRAM memory is needed for this function. That's a maximum of 2048 bytes when using the external character generator mode.

The location and size of each of the three memory areas is user-defined. In other words, you must send the T6963 commands to set up pointers to the three regions during the initialization phase of your program.

The T6963 controller is interfaced to the host via an 8-bit parallel interface, using two registers, command/status and data. To access the raster SRAM using an interface such as this, you must first issue a command that loads the desired memory address. Then you must issue a command that reads or writes to that address.

This requires quite a few data transfers just to read or write one memory byte. To speed up the process for sequential memory accesses, there are two alternatives. There are data read/write commands that automatically increment the SRAM memory pointer every time they are executed, which eliminates the need to send a new memory pointer for each memory access.

And, for large, sequential accesses to the SRAM memory, there is the data auto mode. After the data auto write command has been sent, you write a continuous stream of data to the T6963 data register, which then loads it into sequential SRAM locations. Alternatively, using the data auto read command followed by a repeated reading of the data register will return contiguous SRAM memory data. To exit this mode, there is the data auto reset command.

Text is placed on the screen by calculating the proper text memory address to correspond to the desired row and column and then writing the desired character code to this address. The only caveat is that the character codes that Toshiba uses are not the ASCII character codes; instead, the T6963 uses the ASCII code minus 32. Because the first 32 ASCII character codes are not displayable control characters, there is logic to this decision. It frees 32 character codes in the upper 32 locations of the ASCII 7-bit character set.

Toshiba uses this area for accented vowels and currency figures that are useful in an international context. It would have made more sense to place these international characters in the lower 32 ASCII code locations (in

Pin number	Symbol	Function
1	FGND	Frame ground (0 V)
2	V_{SS}	Ground
3	V_{DD}	Logic supply (5 V)
4	V_{EE}	Negative supply for LCD
5	–WR	Data write
6	–RD	Data read
7	–CE	Chip enable
8	C/–D	H—command/status L—data
9	A	Backlight anode
10	–RESET	Controller reset
11	D0	Data bit 0
12	D1	
13	D2	
14	D3	
15	D4	
16	D5	
17	D6	
18	D7	Data bit 7
19	FS	Font select H—6 × 8, L—8 × 8
20	K	Backlight cathode

Table 1—*The T6963 controller uses Intel-style interface signals. Note that the LCD backlight signals may differ on other displays with different or no backlight.*

place of the control characters), thus leaving the commonly used characters accessible by using their proper ASCII codes.

An additional enhancement is available when you select the text attribute mode using the mode command, which I'll describe later. In this case, you lose graphics capabilities, but can write various bytes to the graphics bitplane to modify the way each text character is displayed.

Text cursor control is flexible, but must be handled entirely by your firmware routines. You can define eight different cursors from a simple underline to a full block, turn on/off the cursor, or make it blink, all under program control. Unlike those alphanumeric LCD modules you've probably used, the T6963 does not move its cursor along with the text while you place it on the screen. When you write a routine that places an ASCII text string on the screen, you may want to include some code that places the cursor to the right of the last character displayed.

There are two methods of placing graphics data on the screen. For bitmapped graphics, you can transfer the bitmap data directly to the graphics area of the T6963 SRAM. This is straightforward; the T6963's graphics format is similar to that used by the Windows BMP image format. I'll cover this in more detail later.

The second method, the bit set/reset command, places individual points on the screen and draws lines. To use this command, you basically convert the x, y (i.e., Cartesian) coordinate of that point into the proper graphics memory location, then use the appropriate bit set/reset command to access the desired pixel within the 8-pixel range represented by that memory location.

The T6963 is a slow CMOS device and it must constantly refresh the multiplexed LCD screen (which is connected through several ASIC devices). For this reason, it is not always ready to accept commands or data through its 8-bit parallel interface. To address this, the T6963 supports a status register, which informs the host of its readiness to accept further commands or data.

The rule is: check the status register and wait until the controller is ready before sending a command or data byte to the T6963. The only complication is that both bits 0 and 1 of the register must equal 1 in order to proceed with any command or data transfer except when using the data auto mode. In the latter case, bit 2 must equal 1 for data reads and bit 3 must equal 1 for data writes.

Alphanumeric LCD panels have similar timing restraints. When using them, I often employed software delay routines in lieu of the status checking routine to reduce the number of I/O lines needed (i.e., the –RD line) and to eliminate the need to switch the data direction of the port used by the host as the data bus. According to Toshiba, you don't want to try that scheme on this controller! In fact, to reinforce that rule, Toshiba does not indicate the amount of time to perform any of the commands.

T6963-BASED LCD PANEL

The T6963 LCD controller chip can handle LCD panels up to 640 horizontal pixels by 256 vertical pixels. I chose AZ Displays' AGM2464D panel (240 × 64 pixels) because it is reasonably priced and supported by good documentation in PDF format on the company web site. The following discussion applies to the T6963 controller as it is configured in Figure 1.

If you're doing prototypes or small production runs, it's important to note that this display is easily connected to your target board via a 20-pin ribbon cable terminated in a 10 × 2 IDC header. The unit contains a convenient LED backlight that requires only the usual 5 V at about 220 mA. Units that use an electroluminescent (EL) backlight require less power, but need a 600- to 800-V inverter circuit.

The T6963 controller onboard the AGM2464D LCD panel interfaces to the micro via an 8-bit data bus using Intel standard control signals *WR, *RD, and *CS and register select line C/–D. Table 1 shows the pinout of the interface connector. The negative reset line may be left unconnected, because the display has an onboard power reset circuit.

The interface timing is shown in Figure 2. Note that the T6963 has a 150-ns read access time, which is slower than the bus timing requirements of many microcontrollers available today. I experienced intermittent problems when I tried driving the T6963 using the Atmel 90S8515's (with a 7.37-MHz clock) extended SRAM bus mode with wait states enabled. When I compared

Item	Item	Condition	Min.	Max.	Unit
C/*D setup time	t_{CDS}	Figure	100	_	ns
C/*D hold time	t_{CDH}	Figure	10	_	ns
*CE,*RD,*WR clock width	t_{CP}, t_{RP}, t_{WP}	Figure	80	_	ns
Data setup time	t_{DS}	Figure	80	–	ns
Data hold time	t_{DH}	Figure	40	_	ns
Access time	t_{ACC}	Figure	_	150	ns
Data output hold time	t_{OH}	Figure	10	50	ns

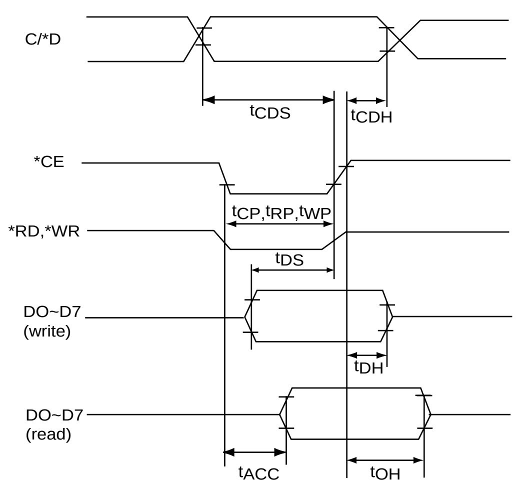


Figure 2—*If you are using a fast microcontroller, you should note the access time specification of the T6963 controller.*

Atmel's bus timing diagram to the T6963, apparently there wasn't a timing violation. But I experienced sporadic data errors. When using this mode, I couldn't connect my oscilloscope's 10× probe to the *WR line without disturbing the data transfers!

For this reason, I recommend driving the T6963 using direct port I/O (i.e., one 8-bit port for the data bus and several lines of another port for the control lines). In this configuration, the *CS line can be tied to ground. Conveniently, because the T6963 is slow, using this slower method of I/O access does not impose a performance penalty.

Step	Command description	Parameter 1	Parameter 2	Command byte
1	Set graphics home address	LSB (0)	MSB (0)	0x42
2	Set graphics area	N (0x20)	0 (0)	0x43
3	Set text home address	LSB (0)	MSB (0x08)	0x40
4	Set text area	N (0x20)	0 (0)	0x41
5	Set offset register (for CGRAM)	M (02)	0	0x22

Table 2—*This table outlines some of the commands used to initialize the controller. The numbers in parentheses in the parameter columns are the specific values that I used.*

SELECTING FONT SIZE

The T6963 controller chip has two font select lines, FS0 and FS1. These are character width select lines, as there is only one font built into the T6963's CGROM and it uses a 5 × 7 matrix for the actual character pattern. However, by changing the levels on the FS0 and FS1 lines, you can change the character spacing to four different widths of five, six, seven, or eight. Note that the character size is always eight pixels vertically.

On the AGM2464D, the FS0 line is tied low, reducing the choice of widths to either six or eight. The remaining FS1 line is brought out to the interface connector as the font select line. When tied to V_{CC} it produces a character width of six and has a width of eight when grounded.

It's tempting to choose the character width of six because it closely matches the 5 × 7 character font and results in a 40-character-wide display. However, there's a side effect when using the graphics display functions. In this case, the graphics pixels are organized horizontally in groups of six on the screen. When displaying bitmaps (and to a lesser extent, drawing lines), the required algorithms are slower and complex because of this mismatch with the 8-bit byte size. For this reason, I configured the display for a character width of eight pixels.

THAT PESKY V_{EE} REQUIREMENT

Unlike those alphanumeric LCD panels that you've probably used, most graphics panels, including this one, require both a 5-V (V_{CC}) and a negative (V_{EE}) supply. The negative power supply must have several important characteristics. It must be stable, but adjustable over a range from approximately –8 to –14 V.

Achieving optimum contrast requires a different V_{EE} at different ambient temperatures, so it is rather important to be able to adjust V_{EE} using an accessible pot or software commands. The V_{EE} current requirement is less than 3 mA.

Before you hook up your new LCD panel to your bench supply, be aware that power supply sequencing is important to prevent damage to the LCD. For this particular unit, it simply means that V_{EE} must be kept off for at least 20 ms after the V_{CC} supply comes up and must disappear before the V_{CC} supply shuts down. Because the LCD unit draws little current from the negative supply, using a multi-output bench supply almost guarantees that the sequencing requirement will be violated, causing the LCD to be damaged. I didn't take any chances, and designed a V_{EE} supply based on a MAX749 digitally-adjustable LCD bias supply.

INITIALIZING THE CONTROLLER

When the T6963 is powered, it is not configured and the LCD panel attached will display nothing. Even after the controller has been properly initialized, the LCD panel still will display garbage until the text or graphics areas of the RAM is cleared.

When testing your new LCD panel, it makes sense to start by executing the initialization code without the screen clear routines. And then use the resultant garbage-filled screen to adjust V_{EE} for the best contrast. There is a narrow range of V_{EE} that produces a readable display.

As mentioned earlier, the T6963's status must be checked prior to writing commands or data. Therefore, in all of the following command sequences, it is assumed that the status checking routine precedes each byte sent. With that in mind, the data write and command write

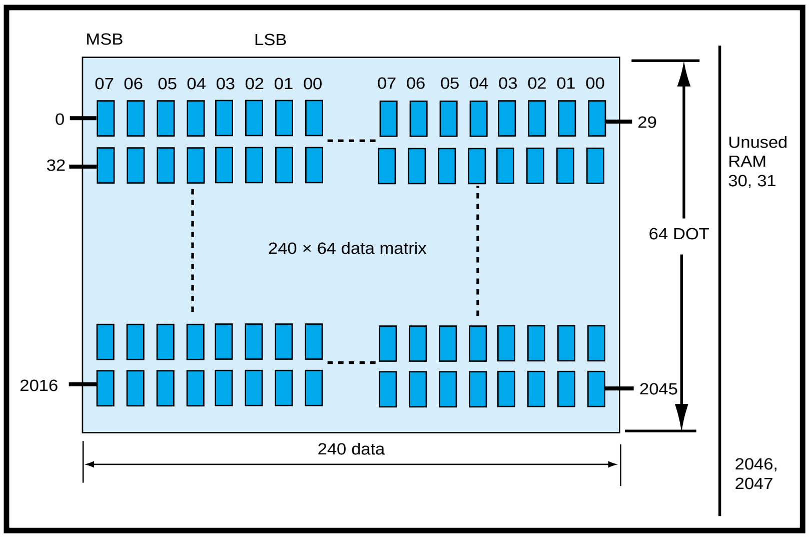


Figure 3—*If you follow my initialization recommendations, the graphics memory organization will look like this.*

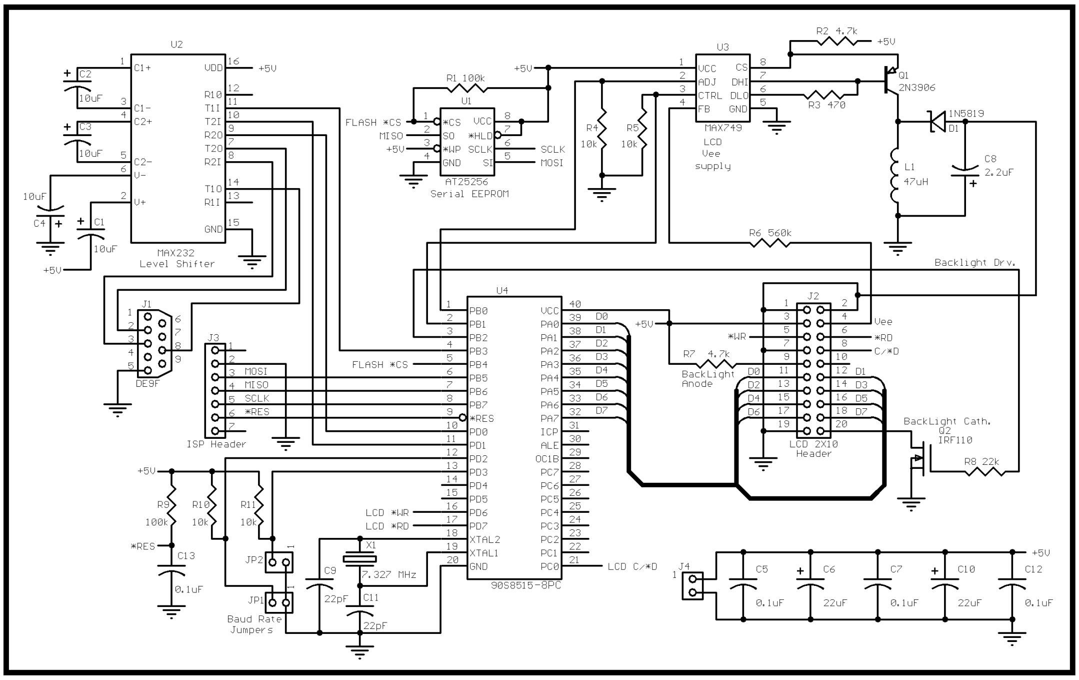


Figure 4—*The serial backpack comprises only four ICs, including the optional serial EEPROM, which stores only bitmap images.*

routines that I wrote include the status checking code at the start of each of those two routines.

The T6963 expects commands to be sent to it with the data parameter byte(s) preceding the actual command byte. Commands require 0 to 2 parameter bytes. In the following commands, the parameters must be sent to the T6963's data register and the command (last byte) must be sent to the command register.

Now, I'll outline the procedure needed to initialize the T6963. If you want to read more detail, check out the T6963 datasheet and a detailed T6963 application note available on my web site.

INITIALIZATION DETAILS

The first issue is setting the graphics home address. Because graphics operations require the most calculations, it makes sense to place the graphics home address at RAM address 0. That eliminates the need to add an offset for graphics operations (see Table 2).

Setting the graphics area is the next issue. Because the T6963 can control many different sizes of LCD panels, you have to tell it how many RAM bytes are needed for each horizontal line. For a 240-pixel display, this is 30. As I'll explain later, it makes sense to use a value that is the next largest binary number (32). Although this wastes some RAM space, it's moot because the 8 KB used in the AGM2464D is twice as big as needed.

Then, you want to set the text home address. You can place the text RAM anywhere, but I choose to place it directly after the graphics RAM. The graphics RAM is 32 bytes × 64 lines, or 2048 bytes, so the text RAM can be placed at 0x0800. Check out Table 2 for the details.

Setting the text area is the next step. Like the graphics area, the text area must reflect the LCD panel size. For a given panel size, the number of characters displayed per line depends on the font width selected. For example, using a 240-pixel display and a character width of 8 (as I configured the AGM2464D), the number of characters per line is 30. Set the text area (designated N in Table 2) to be equal to or larger than this number.

Similar to the graphics area set command, it's advantageous to use the next largest binary number (32). Note again, this wastes 2 bytes of RAM per line, but the routines that convert the text row and column into a text RAM address are simplified.

Now, let's talk about setting the display mode. This command sets up the way the controller handles several features. There are individual text and graphics bitplanes, and you must select which one (or both) will be used. This command also controls whether or not the text cursor is visible and whether or not it will blink (see Table 3).

Table 4 states another command that sets up other aspects of the display mode not covered by the preceding command. I use the value 0x80, which enables both text and graphics bitplanes in OR mode, and uses the internal CGROM for the first 128 text characters. The OR mode simply means that for a given screen location, if either a text pixel or graphics pixel has a value of 1, then the screen will be black at that location.

The set offset register command (start address of CGRAM) is optional, needed only if you are defining custom characters. The external CGRAM mode requires 8 bytes per character (2048 bytes total). Place the CGRAM starting address at some location in RAM that will not interfere with the text and graphics RAM area. This address is expressed differently than that used by both the text and graphics start setup commands. The value M, shown in Table 2, is as follows:

M = 0 for 0x0000 start
M = 1 for 0x0800
M = 2 for 0x1000

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	1	GRPH	TEXT	CURS	BLNK

If GRPH is 1, graphics bit plane is active
If TEXT is 1, text bit plane is active
If CURS is 1, cursor is visible
If BLNK is 1, cursor will blink (if visible)

Table 3—*For the display mode command, I use the value of 0x9E to enable text, graphics, and a non-blink text cursor.*

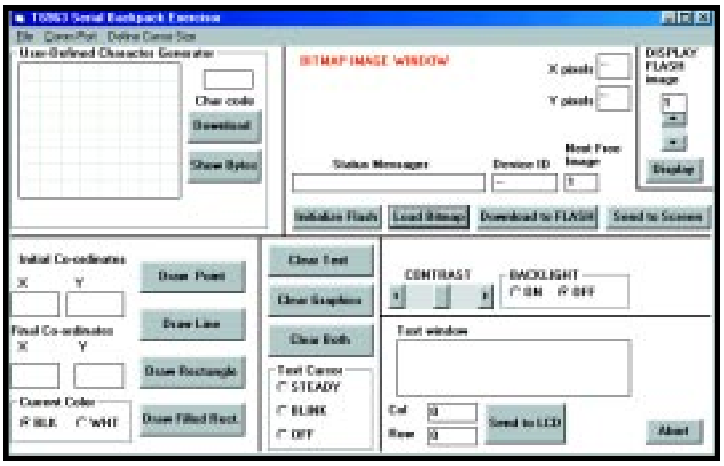


Photo 1—*I developed a Visual Basic program to test the serial backpack. It also downloads bitmap images to the backpack's serial EEPROM.*

M = 3 for 0x1800

and so on up to the size of the RAM fitted on the LCD panel.

After the preceding initialization has been done, the LCD screen will be active and filled with random text characters superimposed on random graphics patterns. To clear the screen, fill both the text and graphics RAM areas with zeros. This is the perfect place to use the data auto mode. Check out the steps of the graphics clear screen example (with graphics screen at RAM address 0).

- set address pointer to graphics RAM start (0x0, 0x0, 0x24)
- set data auto mode on (0xB0)
- execute a loop that sends 2048 zeros to the T6963's data register
- exit data auto mode (0xB2)

Clearing the text screen is similar. Set the address pointer to the text RAM start address and write 256 zeroes to the data register.

BASIC GRAPHICS OPERATIONS

Graphics operations can be broken down into two categories, vector operations (drawing points and lines) and displaying bitmap images. Let's look at vector operations first. Because the basis for all vector operations is the placement of a point on the screen, I'll start there.

A common operation for all vector operations involves converting an x, y coordinate(s) into an offset in the graphics RAM memory space. Figure 3 shows the graphics screen map when using a 240 × 64 LCD and the values recommended previously.

The offset into graphics RAM is (x\8) + 32 × y, where \ denotes an integer divide. Because you want fast graphic operations, you can speed up this conversion by substituting inherently fast shift operations for the slower divide and multiply operations. Thus, the offset into graphics RAM is (x SHR × 3) + y SHL × 5, where *x SHR × 3* denotes three right shifts on *x* (you may use an 8-bit shift routine) and *y SHL × 5* denotes five left shifts on *y* (you must use a 16-bit shift routine).

You use this offset with the set address pointer command to point to the proper RAM location. The command structure is LSB, MSB, 0x24, where LSB and MSB are the lower and upper bytes, respectively, of the offset just calculated.

Now that you're pointing at the correct location in graphics RAM, you must determine which bit at that location must be set or reset and issue the correct bit set/reset command. To place a dot, use the following command:

bit set/reset = 0xF8 + ([255 – x] AND 7)

To erase a dot, use:

bit set/reset = 0xF0 + ([255 – x] AND 7)

DRAWING LINES

Until you've actually written the code, it's easy to assume that drawing lines wouldn't be more difficult than cobbling together some loops using the routine for placing points described earlier. This is true only for two special cases—vertical and horizontal lines. Angled lines require more attention.

I'm not a mathematician. I'm sure there have been many algorithms developed to break down a line into a series of points. However, knowing the way that the T6963 controller organizes its graphics memory and the best way to use the Atmel AT90S8515's instruction set, I designed my own routines that work efficiently.

The line draw routine passes through the x, y coordinates of the endpoints of the desired line. First the endpoints are checked to see if either a horizontal or vertical line is being specified; if so, jump to dedicated routines, which are faster than the ones for angled lines.

For angled lines, the routine used will depend on the angle of the line. In theory, you should divide the 360° into eight 45° zones. These zones can be reduced to four if you allow the start and end point of a line to be swapped in certain cases. This is reasonable because you don't care whether a line is drawn from start to end or vice versa. It draws so quickly that the end effect is visually equivalent anyway.

Determining which zone the line falls into is accomplished by comparing the absolute value of Δx with the absolute value of Δy. If |Δx| is greater, then the proper quadrant is determined by whether y_0 or y_1 is greater. If, on the other hand, |Δy| is greater, then the proper quadrant depends on whether x_0 or x_1 is greater.

Based on these four quadrants, there are four specialized line drawing routines:

$\Delta x > \Delta y$ and $x_1 > x_0$ and $y_1 > y_0$
$\Delta x > \Delta y$ and $x_1 > x_0$ and $y_0 > y_1$
$\Delta y > \Delta x$ and $x_1 > x_0$ and $y_1 > y_0$
$\Delta y > \Delta x$ and $x_0 > x_1$ and $y_1 > y_0$

The first and second cases have a larger *Δx*. For the first case, calculate $^{\Delta y}/_{\Delta x}$ and then step through all x integer values from x_0 to x_1 while adding this fraction to the *y* value each time. Do the same for the second case, except substract the fraction.

In the latter cases, which have a larger *Δy*, calculate $^{\Delta x}/_{\Delta y}$ and then step through all *y* integer values from y_0 to

y_1, while adding (for the third case) this fraction to the x value each time. For the fourth case, subtract the fraction.

In practice, the routines have to be more complex to produce accurate angled lines. You must add a bias of half the value of the fraction when calculating the coordinates of the second point drawn. I used an integer divide routine with scaling instead of floating-point routines to calculate the value of the fraction. Because of the routine I chose, I had to add the remainder of the division operation to the value calculated for the last point of the line.

The speed of these routines is so fast that the Atmel microcontroller is waiting for the T6963 controller to return an unbusy status byte for a significant portion of time. That is to say, the T6963 is what limits the ultimate line drawing speed. Actually, it takes the liquid crystal panel about 100 ms to turn on a pixel and about 250 ms for it to extinguish, so the T6963 is not the bottleneck.

Photo 2—*My prototype was built on a DonTronics SimmStick protoboard, shown plugged into the matching DT003 motherboard.*

The line drawing routines used by some serially controlled, intelligent LCD panels only erase a line properly if you specify the starting and final coordinates in the same order in which they were specified in the corresponding draw command. For example, a line drawn from 0, 0 to 99, 50 may not be erased properly if you pass the coordinates 99, 50 to 0, 0 to the erase routine. My line drawing algorithms do not suffer this shortcoming.

DISPLAYING BITMAPPED IMAGES

After text and vector graphics, the next most common display task is bitmapped graphics. The T6963's graphics RAM organization lends itself nicely to this, because it is similar to the format used by Windows BMP images. Although you can not take a BMP file from a PC and stream it directly into the T6963's graphics RAM space, there is not much preprocessing necessary.

Table 5 gives the locations in a BMP image file where the parameters relevant to B/W images are located. The format of the bitmap array stored in the BMP file differs from what is

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	CG	MC2	MD1	MD0

CG = 0, internal ROM character generator
1, external RAM character generator

MD2	MD1	MD0	
0	0	0	OR mode
0	0	1	XOR mode
0	1	0	AND mode
1	0	0	Text attribute mode

Table 4—*Here's the structure of the mode command.*

needed by the T6963 in the following three ways.

First, BMP files store image data from left to right in groups of 4 bytes (long integers). For example, in an image that is 42-pixels wide, 5 bytes would be needed, but an additional three 0 bytes would be appended to the datastream to meet the 4-byte rule mentioned earlier. You have to weed out these extra 0 bytes when they occur in the BMP file.

Second, BMP files store data from the bottom of the image to the top, whereas the T6963's graphics coordinate system has the origin at the top of the screen.

The third difference is that BMP files use a bit value of 0 to denote black, and the T6963 uses a bit value of 1 for a darkened pixel.

Considering these issues, the data from the bitmap file simply can be transferred to the T6963's graphics RAM directly. The quickest way to accomplish this is to set the T6963's address pointer to the correct location, enter data auto write mode, and send one complete horizontal line of the BMP file data to the T6963, one byte at a time. Then, add 32 to the address calculation performed earlier and send another line until the entire image is transferred. When finished, turn off the data auto mode by issuing the data auto reset command.

And, there's a lot more to the BMP file format. Microsoft publishes the full specifications of the format.

SERIAL BACKPACK

The background for this article was gathered while designing an instrument that required a graphics display. I first bought a commercial LCD that had a serial data interface and an intelligent controller. Although expensive, it was easy to use because of the serial interface and high-level commands.

What I needed that the commercial unit lacked, was a function best described as an oscilloscope display mode. In other words, I wanted to be able to send a stream of amplitude values to the LCD that would be displayed quickly from left to right. Then, without doing a full screen erase (which would result in a dim image with an annoying flicker), I wanted to be able to refresh the image with new data.

Doing this with commercial units involved keeping track of the old data, sending this data and a command to erase it one point at a time, then writing new data to the display. The amount of data that had to be transferred over the serial data link was significant and the whole process was slow. Additionally, the host had to keep two complete sets of data, which required more RAM than was available in my microcontroller.

To solve this problem, I designed my own serial backpack that works with commonly available T6963-based LCD panels such as AZ Display's AGM2464D or Optrex's DMF5005. This serial backpack performs the following functions.

For the text, the serial backpack does cursor placement and attributes, writes ASCII string to the LCD, clears the text screen, defines custom font characters, and adjusts LCD contrast and backlight. For vector graphics, it draws and erases points, lines, rectangles, and blocks.

The serial backpack performs bitmap graphics, too. It clears the graphics screen, downloads the bitmap image to flash EEPROM, displays the bitmap image from EEPROM, and displays the bitmap image from serial data input (see Photo 1). More of its functions include displaying and refreshing a dynamic x, y coordinate data array.

Although the use of meaningful

Offset	Size	Type	Description of the variable
1	2	Character	"BM" to indicate that the file is a bitmap
11	4	Long int.	Offset into file of start of bitmap data
19	4	Long int.	Width of the image in pixels
23	4	Long int.	Height of the image in pixels

Table 5—*These BMP file parameters determine the size of the image and locate the position of the bitmap in the file.*

commands and ASCII strings for parameters make it easy when using a high-level language, it results in longer command/datastreams. Because this device gets its commands/data through a serial data link, I designed a compact command/parameter structure to speed up things.

Figure 4 is a schematic diagram of the backpack. I chose the Atmel 90S8515-8PC microcontroller largely because it has 8 KB of ISP flash memory, 512 bytes of RAM, and a fast RISC instruction set. The prototype was built on a small DonTronics SimmStick protocard (see Photo 2). The unit's 5-V power supply and MAX232 reside on a DonTronics DT003 mini-motherboard. The firmware is written entirely in assembly language and uses about half of the available flash memory.

The design also includes an Atmel AT25256 32K × 8 SPI EEPROM. This nonvolatile memory can be used to store up to 127 bitmap images. These images can be quickly referenced by number and displayed using the bitmap display command. Because monochrome bitmap images lend themselves to data compression, I implemented a run-length-encoding algorithm in the EEPROM image downloading routine. It crams five to 10 times as many images into the available EEPROM and is quicker to access a small, encoded image file from the serial EEPROM, even though it takes a few additional CPU cycles to decode the RLE data.

A MAX232 is used for RS-232 level shifting. One section of the device is used for the receive data from the host. The backpack implements both hardware and software handshaking, so two additional sections are needed for the transmit data and CTS signals.

Unfortunately, the MAX232's built-in negative charge pump doesn't generate enough voltage for the LCD's V_{EE} supply. Therefore, the next easiest way to generate the V_{EE} supply for the LCD panel is to use an IC that was designed specifically for that purpose. I chose the Maxim MAX749 digitally adjustable LCD bias supply. This 8-pin DIP plus six discrete components fits the bill nicely. It provides a regulated negative supply that can be set to the appropriate range via a single resistor selection. The voltage then can be adjusted in 64 steps from 33% to 100% of this value, under digital control. This is sufficient resolution to provide good LCD contrast at different temperatures.

The requirement that the V_{EE} supply be sequenced is easily addressed by keeping the MAX749 in shutdown mode for at least 20 ms after the logic (V_{CC}) supply comes up. The two digital control lines used for digital voltage adjustment also are used to place the unit in shutdown mode. At powerup, the firmware in the AT90S8515 microcontroller holds the MAX749 in standby by holding these two digital lines low for the required 20 ms.

The value of R6 was chosen to provide –7.5-V output with the MAX749's internal DAC set at mid-scale (default setting). This voltage provides a reasonable contrast on the LCD panel at nominal room temperatures.

The AGM2464D uses an LED backlight that is controlled by software using a port pin on the micro and an IRFD110 HexFet. The data interface to the LCD panel is via a 2 × 20 header connector. The AGM2464D and Optrex DMF5005 panels use the same pinout, which may be common to many 240 × 64 T6963-based panels.

WRAP UP

If you've considered using a graphics LCD panel but were scared off by the thought of a lot of programming, I hope this article changes your mind. There is a lot of good information on the 'Net, but it takes effort to find it. I posted datasheets from the various manufacturers, application notes, and more. on my web site, as well as the complete documentation package for my serial backpack. ■

Author's Note: I'd like to acknowledge Steve Lawther for his good application note on the T6963 controller.

Brian Millier is an instrumentation engineer in Dalhousie University's chemistry department, Halifax, NS, Canada. He also runs Computer Interface Consultants. You may reach him at brian.millier@dal.ca.

SOFTWARE

The object code is available on the *Circuit Cellar* web site.

SOURCES

AGM2464D
AZ Displays, Inc.
(949) 360-5830
Fax: (949) 360-5839
www.azdisplays.com

DMF5005
Timeline, Inc.
(310) 784-5488
(800) 872-8878
Fax: (310) 784-7590
www.digisys.net/timeline

AT90S8515-8PC
Atmel Corp.
(714) 282-8080
Fax: (714) 282-0500
www.atmel.com

MAX749, MAX232
Maxim Integrated Products
(408) 737-7600
(800) 998-8800
Fax: (408) 737-7194
www.maxim-ic.com

Programmed AT90S8515-8PC for the serial backpack
Brian Millier
Computer Interface Consultants
(902) 494-3709
Fax: (902) 494-1310
bmillier.chem.dal.ca

SimmStick protoboard
DonTronics
Fax: +613 9338 2935
www.dontronics.com

FEATURE ARTICLE

George Novacek

Designing for Reliability, Maintainability, and Safety

Part 1: Getting Started

If you think that designing for reliability and maintainability are just steps in the development process that eat up more budget without adding any value, then you might want to listen up because George has evidence that proves otherwise.

Electronic equipment designers are familiar with the concepts of reliability, maintainability, and safety (R/M). But sadly, we use these disciplines much less than they deserve, especially in consumer product development. Many people find them boring and think they chew up a big chunk of the precious development budget but add no value.

In this article, I want to show you the basic steps in performing R/M analyses and how they do add value to your design. You'll learn to appreciate their benefits and, hopefully, come to the conclusion that your time is well invested using them. Not all manufacturers spend the time to ensure good design, but as you will learn, we could be buying better, safer products at lower prices if they did.

To be effective, the analyses must be performed concurrently and as an integral part of product development. I will take you through development of a hypothetical controller to illustrate the fundamentals of the R/M engineering approach. I set up the project to demonstrate the use of R/M tools, not the design of a controller. So, some design decisions are tailored to exaggerate R/M aspects. For the same reason, some issues have been simplified or omitted.

HAZARD ANALYSIS

Let's assume you are contracted to develop an electronic controller for a hot tub, to maintain its water temperature at 100°F (±2°). It is heated by a gas-fired burner capable of raising the water temperature to the boiling point quickly. The hot tub supplier (your customer) should start the project by performing a system hazard analysis. And, you should know enough about the system to prepare your own. The requirements pertinent to the subsystem become a part of the performance specification.

The simple analysis shown in Table 1 has only two potential failures. The failures and their permitted probabilities of occurrence per hour of operation are the result of a trade-off between requirements and cost.

With the possibility of personal injury at hand, I'm not aware of any system where it would be considered an acceptable risk to allow such a design weakness. Performing the hazard analysis and implementing its results reduces risk and shows reasonable care and prudent design in court (just in case).

The probability of 10^{-9} failures per hour of operation generally is accepted as "never." With the exponential fault distribution, which is most popular in electronics and yields a constant failure rate, it represents a 50% chance of experiencing such a failure after about 79,000 years of continuous operation.

The second line of Table 1 shows noncritical subsystem failures when the hot tub is not as hot anymore. You don't want such failures but can live with them. Statistically, all components will fail at some point. As you will learn during this design exercise, decreasing the probability of a failure is expensive. When there is no potential for personal injury, the decision boils down to the manufacturing cost versus potential customer unhappiness, cost of service, warranty, maintainability, life cycle cost, and so on.

Imagine that the controller fails. It doesn't matter why, now you have a customer complaint and must send out a maintenance technician. If it happens during the warranty period, you pay for the repair. With the probability of failure pegged at 10^{-5}/h as a design goal, you must expect a failure every 100,000 h of the controller operation. But, suppose sales take off and during the next few years you sell 10,000 units. If they run 24 h per day, you can expect one failure every 10 h.

Is it possible to lower the failure rate by an order of magnitude? One million hours of mean time between failures (MTBF) would drastically reduce the service cost but will be expensive to achieve. What is the cost of improving MTBF compared to the savings in maintenance and warranty cost? Will there be enough parts to service the equipment several years from now, recognizing that the current life cycle of microelectronic parts is merely five years or less?

Let's say you must extend a five-year warranty. Based on the probable failure rate, you can estimate the warranty cost. How will it affect profitability? R/M analyses also help make these business decisions. Provided they are performed concurrently with the design, their results are implemented in a closed-loop system for optimal results. Discovering the laws of statistics after the product introduction to the market may be revealing, but usually too late.

RELIABILITY FUNDAMENTALS

To fully appreciate the aspects of reliability, you need to review the fundamentals of failure prediction. Because electronic components can be most often modeled by constant failure rate (λ), which is the characteristic property of exponential failure distribution, you won't have to go into the gory details of statistical analysis. The mathematics will be straightforward.

Suppose a sample population of a component you are interested in is tested while you record observed failures, plotting them against time of their occurrence. You can plot the number of occurrences within given, short, time intervals to obtain a frequency distribution plot. Or, you can record the cumulative number of failure occurrences against the time as you proceed with the test (cumulative frequency distribution).

The cumulative frequency distribution of the majority of electronic components will be exponential and resembles the curves in Figure 1. The mathematical model for the frequency distribution is called probability density function (PDF) and for exponential distribution is expressed as $f(t) = \lambda \times e^{-\lambda t}$. The cumulative frequency of distribution is modeled by the cumulative distribution function (CDF):

$$F(t) = \int_0^t f(y)dy$$
$$0 \le t \le \infty$$

where *f(y)dy* is a dummy variable of integration. For the exponential distribution, you can write the following functions for PDF and CDF, respectively:

$$f(t) = \lambda \times e^{-\lambda t}$$

$$F(t) = 1 - e^{-\lambda t}$$

where λ is a single unknown that defines the fault distribution. You can calculate reliability (the number of surviving units) as:

$$R(t) = 1 - F(t)$$

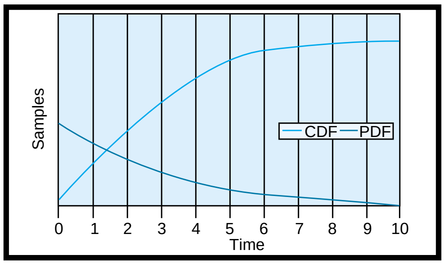


Figure 1—*Probability density function (PDF) and cumulative distribution function (CDF) are the results of exponential failure distribution in components and are characteristic of the constant failure rate typical of electronic parts.*

Then, use this to arrive at the expression for an instantaneous failure rate:

$$h(t) = \frac{f(t)}{R(t)}$$

and solve for constant λ:

$$h(t) = \frac{\lambda \times e^{-\lambda t}}{e^{-\lambda t}} = \lambda$$

A component following an exponential life distribution exhibits the same probability of failure in the next hour regardless of whether it is new or used. It does not age or degrade with use. Failure occurs at a constant rate, unrelated to the hours of use.

This important, seemingly illogical concept allows you to gain equivalent information from testing 10 units for 10,000 h as if from 1,000 units for 100 h. It also means that the "impossible" 10^{-9} failure is as likely to happen in the first 5 min. of operation as 114,000 years from now.

If, based on observation, you suspect that the failure rate does depend on the time used, it may be because of wear caused by improper derating or the exponential fault distribution does not apply for this part.

A measure of reliability more commonly used at the user level for irreparable parts is mean time to failure (MTTF). For components with exponential life distribution, this is a reciprocal of λ. For assemblies where a failed component can be replaced, MTBF is appropriate. For exponential fault, distribution also equals $1/\lambda$.

Figure 2 shows the well-known bathtub diagram. The diagram consists of three

Failure description	Failure effect	Maximum probability
The controller fails to turn off the burner when the water reaches 102°F.	A critical failure must not happen under any circumstances, personal injury may result.	$<10^{-9}$
The controller fails to turn on the burner when the water temperature drops below 98°F.	During a noncritical failure, the tub is not useable and the system is no longer available. Customer dissatisfaction results.	$<10^{-5}$

Table 1—*Even a simple hazard analysis matrix is an effective tool for identifying system weaknesses and potential hazards that the design must eliminate.*

Reliability Models For Electronic Components

The calculated value is λ_p, which represents the predicted number of failures per 10^6 hours. The reliability model for microelectronic circuits (ICs) states that:

$$\lambda_p = (C_1 \times \pi_T + C_2 \times \pi_E) \times \pi_Q \times \pi_L$$

where:

C_1 = die complexity; 0.14 for the PIC controller and 0.020 for the regulator 7805.

π_T = temperature coefficient. Assuming the junction temperature $T_j < 100°C$ for both ICs, it will be 1.5 for the PIC controller and 16 for the regulator.

C_2 = a constant based on the number of pins. 0.0034 is used for the PIC with 8 pins and 0.0012 for the 3-pin regulator.

π_E = environmental constant. Assume the equipment will operate in a "ground fixed" environment, a benign location with average ambient temperature of 25°C, not exceeding 45°C.

π_L = learning factor; 1 for ICs more than two years in production.

π_Q = quality factor. This is the most controversial coefficient. For military screened components it is between 1 and 2, but climbs to 10 for commercial components. Many critics have established that the penalty for commercial, off-the-shelf parts is unrealistically high, especially when taking into account modern manufacturing processes.

For diodes, the equation looks like:

$$\lambda_P = \lambda_b \times \pi_T \times \pi_T \times \pi_S \times \pi_C \times \pi_Q \times \pi_E$$

where:

λ_b = base failure probability related to the construction; 0.0012 for switching and general-purpose diodes, 0.0030 for power rectifiers, and 0.0013 for transzorbs.

π_T = temperature coefficient; 3.9 for junction temperature $T_j <$ 70°C.

π_S = is based on stress 1.0 for transzorbs and 0.054 for other diodes in the system, provided they are not exposed to more than 30% of their rated characteristics.

π_C = contact construction factor; 1.

π_Q = 8.0 for plastic encapsulated devices.

π_E = environmental constant; 6.0 for the "ground fixed" environment.

Next is the solenoid driver power MOSFET (less than 1-W dissipation):

$$\lambda_P = \lambda_b \times \pi_T \times \pi_A \times \pi_Q \times \pi_E$$

where:

λ_b = base failure probability related to the construction; 0.012 for power MOSFET.

π_T = temperature coefficient; 2.3 for junction temperature $T_j <$ 70°C.

π_A = 1.5 for switching applications.

π_Q = 8.0 for plastic encapsulated devices.

π_E = environmental constant; 6.0 for the "ground fixed" environment.

Resistors' reliability calculates as follows:

$$\lambda_P = \lambda_b \times \pi_T \times \pi_P \times \pi_S \times \pi_Q \times \pi_E$$

where:

λ_b = base failure probability related to the construction; 0.0037 for RLR resistors and 0.0019 for thermistors.

π_T = temperature coefficient; 1.3 for resistor operation less than 50°C and 1 for the thermistor.

π_P = is determined by the dissipated power; 0.44 for 100 mW and greater, 1 for 1 W, and 0.44 for the thermistor.

π_S = 1.1 for stress factor 0.4 (i.e., you don't operate the device at more than 40% of its rated characteristics). It equals 1.0 for the thermistor.

π_Q = 3.0 for devices without established reliability.

π_E = environmental constant; 6.0 for the "ground fixed" environment.

Capacitors' reliability is expressed as:

$$\lambda_P = \lambda_b \times \pi_T \times \pi_C \times \pi_V \times \pi_{SR} \times \pi_Q \times \pi_E$$

where:

λ_b = base failure probability related to the construction; 0.00051 for fixed, metallic film capacitors and 0.00040 for tantalum capacitors.

π_T = temperature coefficient; 1.6 for capacitor operation less than 50°C

π_C = a factor for capacitance; 0.81 for the fixed capacitors assumed to be 0.1 µF and 1.6 for the electrolytic capacitors assumed to be 10 µF.

π_V = 1 for stress factor 0.3.

π_{SR} = 1 for both types.

π_Q = 3.0 for devices without established reliability.

π_E = environmental constant; 10.0 for the "ground fixed" environment.

The next device on the list is the transformer:

$$\lambda_P = \lambda_b \times \pi_T \times \pi_Q \times \pi_E$$

where:

λ_b = base failure probability; 0.022 for low-power transformers.

π_T = temperature coefficient; 1.4 for operation less than 50°C.

π_Q = 3.0 for devices without established reliability.

π_E = environmental constant; 6.0 for the "ground fixed" environment.

And finally, for quartz crystals:

$$\lambda_P = \lambda_b \times \pi_Q \times \pi_E$$

where:

λ_b = base failure probability related to the crystal frequency; 0.022 for 10 MHz.

π_Q = 2.1 for nonmilitary devices.

π_E = environmental constant; 3.0 for the "ground fixed" environment.

The results of the calculations are tabulated in Table 2.

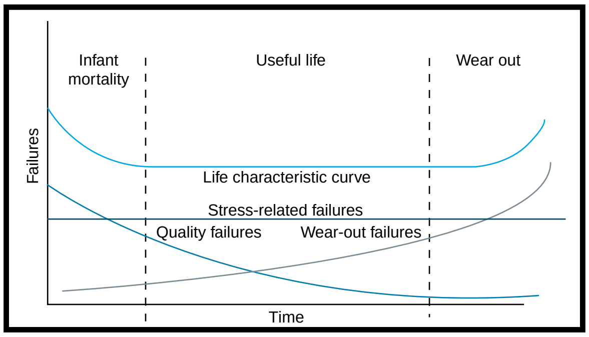


Figure 2—*The product life cycle characteristic curve is composed from the constant failure rate of components, increased at the beginning by infant mortality and at the end by wear. (This figure is not to scale.)*

curves and three distinct areas. The quality curve is predominant during the initial life period and is often referred to as infant mortality, which is the result of design, handling, or workmanship problems.

Infant mortality effects can be reduced by using robust designs and manufacturing process control. At the end of the manufacturing process, a good burn-in or environmental stress screening (ESS) period will weed out the majority of the failures. Products shipped from the factory should be past the infant mortality curve.

The useful life period is characterized by stress-related failures, in other words, the MTTF or MTBF. The infant mortality curve can bottom out within a few days of "shake 'n bake," but the useful life period is counted in years.

Finally, at the end of their life, components begin to fail because of wear. Although common in mechanical systems, well-derated electronic systems seldom reach this period.

RELIABILITY PREDICTION

Reliability prediction analysis results in definition of λ_p, the predicted failure rate, which is expressed as a number of failures per 10^6. This number forms the basis for other R/M analyses. In most electronic systems with a constant failure rate, the MTBF and MTTF are the reciprocals of λ_p stated in hours. If a device is not repairable (e.g., a failed transistor), MTTF is used. If it can be repaired, MTBF is used.

Because they attempt to forecast the future, reliability prediction methods have been the subject of many heated debates. Several schools of thought exist, each having adherents and just as many opponents. This article is not the forum for getting into the thick of the debate. I'll discuss MIL-HDBK-217F because it's a widely recognized model based on the constant failure rate. [1]

Remember that you are extrapolating historical or test data into the future. This data is used as a yardstick for performance evaluation and improvement. So, you need to understand what the numbers mean and must not look at them as the only objective.

Tweaking the numbers will be self-defeating. Field returns are always a more powerful statement of performance than statistical predictions. Both you and the customer need to understand that the failure probability of 10^{-6} is not a guarantee of a million-hour, failure-free operation. A failure can occur in the first or one-millionth hour. On the positive side, reliability models are conservative, the equipment outperforms the statistics.

Let's start with preliminary design of the controller. You'll calculate its predicted reliability, put it through FMECA and FTA analyses, and review the results (see Figure 3).

For this example, I used a CMOS PIC micro operating from a 5-V power supply provided by a three-terminal 7805 regulator. The water temperature is detected by thermistor R3 and the gas-fired water heater is controlled by solenoid valve (SV) L1. The valve contains an internal freewheeling diode to suppress back-EMF kick and is switched off and on by power MOSFET Q1.

Diodes D2 and D4 clamp the analog input of the PIC within 0 to 5 V to protect the micro and prevent ESD damage. One of the built-in ADCs reads the thermistor voltage. The second ADC monitors the current through the solenoid valve to provide short-circuit protection and to monitor operation. The valve is external and supplied by the system integrator, so exclude it from the reliability calculation; although, it is an important player for safety. Tranzorb D5 protects the MOSFET driver against voltage

Component	Description	$l_p/10^6$ hours	MTTF
R1	Small resistor RLR	2.7936E-02	3.5795E+07
R2	Small resistor RLR	1.0794E-02	9.2647E+07
R3	Small resistor RLR	1.0032E-02	9.9681E+07
R4	Small resistor RLR	2.7936E-02	3.5795E+07
R5	Small resistor RLR	1.0794E-02	9.2647E+07
R6	Current sensing resistor	6.3492E-02	1.5750E+07
C1	Tantalum capacitor 10 mF	3.0720E-02	3.2552E+07
C2	Tantalum capacitor 10 mF	3.0720E-02	3.2552E+07
C3	Metallic capacitor 0.1 mF	1.9829E-02	5.0432E+07
C4	Metallic capacitor 0.1 mF	1.9829E-02	5.0432E+07
C5	Metallic capacitor 0.1 mF	1.9829E-02	5.0432E+07
C6	Metallic capacitor 0.1 mF	1.9829E-02	5.0432E+07
C7	Metallic capacitor 0.1 mF	1.9829E-02	5.0432E+07
C8	Metallic capacitor 0.1 mF	1.9829E-02	5.0432E+07
Q1	Power MOS-FET $P_D = 1$ W	1.9872E+00	5.0322E+05
U1	7805 regulator	1.1680E-01	8.5616E+06
U2	PIC12C672 microcontroller	1.8160E-01	5.5066E+06
D1	1A bridge rectifier	1.2131E-02	8.2436E+07
D2	Signal diode (1N914)	1.2131E-03	8.2436E+08
D3	Signal diode (1N914)	1.2131E-03	8.2436E+08
D4	Signal diode (1N914)	1.2131E-03	8.2436E+08
D5	Transzorb	2.4336E-01	4.1091E+06
X1	Quartz crystal	1.3860E-01	7.2150E+06
T1	Transformer 120 V primary	5.5440E-01	1.8038E+06
Controller total		3.5691	280,180 h

Table 2—*Preliminary failure rate calculation also indicates that you are on the right track and that the customer's specification is achievable.*

transients, which could be the result of the freewheeling diode failure.

To calculate the predicted failure rate, you could use one of several expensive programs. Some methods even extract components and their operating conditions out of the schematic capture program, simplifying the chore that generations of engineers have performed manually. My program is a small circuit and performing the calculation by hand will be good exercise. As stated previously, you're using the MIL-HDBK-217F model with exponential distribution, assuming a constant failure rate. All the values and calculations come from that source. Read the "Reliability Models for Electronic Components" sidebar for the details.

Immediately it is apparent that the resulting failure rate and MTBF of nearly 300,000 h satisfy the 10^{-5} system availability requirement the customer defined in the hazard analysis. Can it be improved? Let's take a closer look, because this would minimize future warranty claim costs, maintenance requirements, and improve customer satisfaction.

All components are well derated (i.e., working at less than 30% to 40% of their specified ratings). Further derating will have a minimal effect on their reliability improvement. But, five components have failure rates that are greater than the rest: Q1, U1, U2, D5, and T1. What can you do?

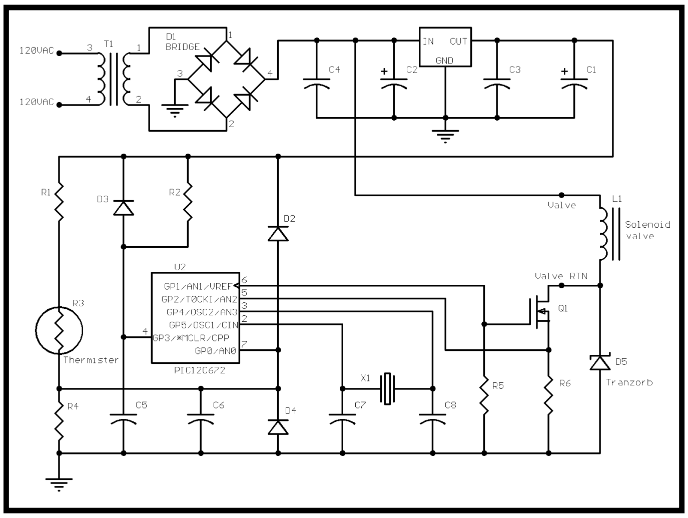


Figure 3—*Here's my first run at the controller schematic diagram. The design is simple, straightforward, and appears to do what is needed.*

Q1, U1, and U2 work at conservatively estimated junction temperature $T_j = 100°C$. Keeping the ambient operating temperature at 27°C and with efficient heat sinking, T_j can be reduced to 50°C. Heat is the reliability killer and even a small reduction will have a significant effect.

Transzorb D5 and diodes D2 and D3 conduct current during infrequent transients only. You can reduce their contribution by applying a duty cycle. Design T1 to run at a lower temperature and you'll improve its reliability.

Implementation of these steps will increase the MTBF to 714,000 h (λ_p = 1.4). And, for the remaining analyses, you will use the results of these calculations to evaluate product safety.

George Novacek has 30 years of experience in circuit design and embedded controllers. He currently is the general manager of Messier-Dowty Electronics, a division of Messier-Dowty International, the world's largest manufacturer of landing-gear systems. You may reach him at gvovacek@nexicom.net.

SOFTWARE

Reliability calculations are available on the *Circuit Cellar* web site.

REFERENCES

[1] U.S. Department of Defense, *Reliability Prediction of Electronic Equipment*, MIL-HDBK-217F, General Policy Series, no. 14T.

S.E.R. subcommittee, *Automotive Electronics Reliability Handbook*, Society of Automotive Engineers, Warrendale, PA, 1987.

P. Tobias and D. Trindale, *Applied Reliability*, Van Nostrand Reinhold, NY, NY, 1986.

U.S. Department of Defense, *Electronic Reliability Design Handbook*, MIL-HDBK-338, General Policy Series, no. 542.

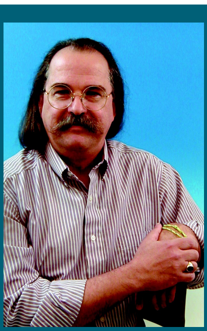
U.S. Department of Defense, *Maintainability Program for Systems and Equipment*, MIL-HDBK-47OB, Washington D.C.: Government Printing Office, 1995.

FROM THE BENCH

Jeff Bachiochi

Sharing Technology with Mother Nature

Out of State with an Internet-Compatible Cell Phone



Out of the office for three weeks meant that Jeff needed a way to stay connected. The recipe for success? Take one new cell phone, one old motorcycle battery….

I find myself spending more and more time on Internet-related matters. I send and receive dozens of e-mails every day (I hate the telephone). I use the instant messenger to keep in contact with my family because it's easier to send a Windows RealPopup message over the home network than to locate someone in the house.

By surfing manufacturers' web sites I get the most up-to-date news and information about parts and products. Oftentimes I also find good application tutorials to help simplify designs. Data books have gone from paper to the web, some without stopping at the intermediate CD stage. I find search engines indispensable for locating both product and educational references. For me, the Internet truly is the information superhighway.

I use my local phone company as my Internet service provider. Because I hardly ever travel out of state, a local number is all I need to connect from home. This summer, a number of events came together in such a way that I was going to be away from home for three weeks straight.

The first week, I was going to Cape Cod, Massachusetts with the family for some much needed R&R (rest and relaxation). The second week, I planned to be in Arizona at Microchip's MASTER conference for some hands-on E&E (experience and education). Finally, it was off to Camp June Norcross Webster in Connecticut for some wilderness C&C (camping and counseling).

All would have been fine if I didn't need to bring up a revised online question and answer forum I was moderating at the time. I was handling it via e-mail, and the new application promised to ease my burden and give the users a real-time feel. The transition period (while I was to be away) required that old and new applications run concurrently, which promised to take much of my time.

Photo 1—*My StarTAC is connected to a Toshiba laptop via Motorola's connectivity kit. Internet Explorer dials out to the cell phone connected as an external modem. My ISP gets me on the Internet where I can surf or connect to get my e-mail.*

Preparing to visit Massachusetts and Arizona was not a big deal; I searched around for a free ISP, which offered local numbers in the areas where I'd be staying. I ended up using freei, which gave me access to the Internet so I could log on to *Circuit Cellar*'s e-mail server on a daily basis. I could also access the FTP site and make daily updates of questions posed to my researchers.

To read and answer e-mails and update the FTP site required one to two hours per day. I found that late evening was the best time for me to accomplish this. The local Hyannis exchange on the Cape was never busy, however, the Phoenix exchange was difficult to get into. The third week was the most challenging.

PREPARATIONS

Our camp wasn't exactly far from civilization, just free from modern hassles. Because the setting lacked power, phone, and running water, free Internet wasn't going to solve the problem of having no utilities.

First, I needed to overcome my loathing of the telephone. In fact, I needed to go a step further, all the way to cellular. Surfing the 'Net for cell phone manufacturers proved to be an empty venture, because few cell phones are Internet-compatible. Of those few that are, one stands out above the rest, the Motorola StarTAC.

The StarTAC 7868W is the dual-mode, analog and digital, dual-band CDMA cell phone with a host of premier features. It operates at 800 and 1900 MHz. Most of you have probably seen this shirt pocket 4.4-oz. handset that provides digital talk and standby times of up to three hours over five days. The optional data connectivity kit coupled with my laptop provides fax capability, Internet access, and e-mail (send and receive).

The StarTAC also has mini-browser capabilities. This works for getting weather information, but the four lines of text definitely won't handle any e-mail. I used my Toshiba Satellite laptop and a real browser for comfortable communication, which would undoubtedly be necessary for extended periods of time.

Average idle	Current measured	Watts
12-VDC source	2.28 A	26.7
15-VDC laptop	1.4 A	21

Table 1—*When my laptop is running off its 110-VAC power supply it requires 21 W. If that supply is plugged into a 12-VDC-to-110-VAC inverter, the 12-VDC battery must supply about 27 W for both items.*

Looking at the talk and standby times of the StarTAC, I expected to get almost two days worth of work (based on two hours a day) before the phone would need recharging. My laptop won't even make that. Something had to give because there aren't any outlets on trees.

The cell phone has a car adapter for charging, but I'd be miles from any cars. Fortunately, I just replaced my motorcycle battery and haven't brought the old one to the recycling center yet. There wasn't enough energy in the failing plates to start an engine, but plenty for what I needed.

A sealed lead acid battery is much safer to use in this situation! The corrosive electrolyte could leak from the unsealed unit if you tip the battery on its side. Lead acid batteries are capable of high currents and, if they are shorted, can cause serious damage by explosion. Use breathable batteries with extreme caution.

I designed a carrier for the battery with a wide base that would keep it more stable and give me a place to mount some components. The first thing I did was wire up a fuse to the battery. Radio Shack has a good assortment of fuses and cigarette lighter accessories, including sockets and plugs. Mounting a cigarette lighter socket onto the battery carrier made connecting to the battery safe and convenient. I measured about 25-mA charging current for the cell phone, which could receive a full charge overnight. This calculates to about 3 Wh per charge.

DC/AC/DC

My Toshiba laptop, on the other hand, is not the perfect match. It cannot be directly powered or charged from a 12-V battery. Its AC plug-in power supply creates 15 VDC for the laptop's DC input. The simplest way to convert my 12-VDC source into power for the laptop would be to use a vehicle power inverter, which produces 110 VAC, and then use the laptop's own power supply. Although simple, it seems inefficient (see Tables 1 and 2).

With the laptop on and idling, it draws ~21 W of power. The power conversion circuitry requires 26.7 W and the idle efficiency is ~78%. With the power to the laptop off, the charging current is ~0.5 A (~7.5 W at 15 V). The power conversion circuitry requires 11 W. The charging efficiency is 68%. We're only talking about a handful of watts here. Is efficiency really important?

If you paid attention last month when I discussed photovoltaic cells, you'll remember that the panel output about 3 W in direct sunlight. If I wanted to use my laptop for two hours each day, I would require 42 Wh of power. That's 14 h of solar panel collection (100% efficiency). The efficiency of the converter would require 53.4 Wh, or 18 h of solar panel collection. I don't know about you, but, I don't get 18 h of sunlight in one day. So, how much power can be saved by directly converting 12 VDC to 15 VDC?

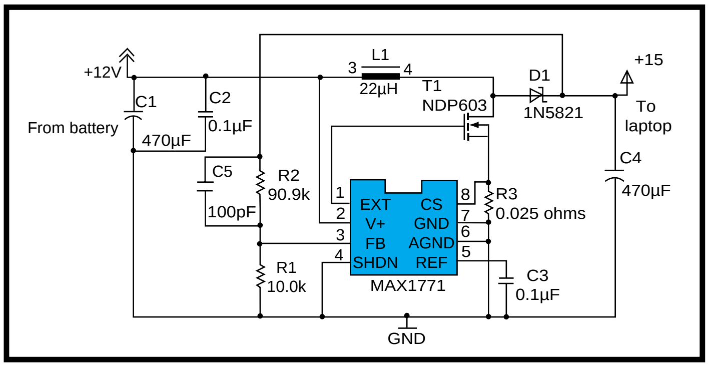


Figure 1—*The DC/DC converter boosts the 12 V from my motorcycle battery up to the 15 V needed to run or charge my laptop. Efficiency is ~95%, as opposed to ~68% using an off-the-shelf DC/AC inverter and the laptop's AC/DC external power supply.*

DC/DC

A boost-mode switching regulator could provide the 15 VDC necessary to run my laptop from the 12-VDC battery. Maxim's MAX1771 looked like it could do the job for me. This device has a fixed 12-VDC output or can be adjusted using a resistor divider on the output (see Figure 1).

Coil current through L1 is allowed to flow when the external MOSFET is enabled by the '1771. This current ramps up, generating a magnetic flux in the coil's core until either the current reaches a maximum (set via sense resistor R3) or the ON-oneshot times out (~16 µs). At this point, an OFF-oneshot triggers and the external MOSFET remains off for ~2.3 µs. Current no longer flows through the coil because the MOSFET now looks like an open circuit. Therefore, the coil's collapsing magnetic field causes the voltage on the anode of the diode D1 (1N5821) to rise until the current can flow through it, charging the output capacitor.

To set the output voltage, a resistor divider combination is selected to produce 1.5 V at the feedback input to the '1771. When the output voltage dips below the nominal level, so does the feedback input. An internal comparator triggers the ON-oneshot again and the cycle starts over. The '1771 differs from the standard pulse width modulation (PWM) circuits in that it adds pulse frequency modulation (PFM) and dramatically reduces the operating current.

With the DC/DC converter in Figure 1 attached to my motorcycle battery, I connected four 1-Ω, 10-W resistors as a load. The battery's power was measured at 36 W and the load current was 34 W (see Table 3). The efficiency of this circuit is ~94%. Charts in the Maxim datasheet suggest that this is about right. Better efficiencies are where the V_{IN} is closest to the V_{OUT}. However, even with lower V_{IN} (3 V), circuit efficiencies can reach over 85%.

DC/DC	Current measured	Watts/h
12-VDC source	3 A	36
16-VDC load	2.1 A	34

Table 3—*Using the circuit from Figure 1, a 4-Ω load has a measured current of 21 A. Notice the high efficiency.*

Charging only	Current measured	Watts
12-VDC source	0.92	11
15-VDC laptop	0.5	7.5

Table 2—*When my laptop is off and charging, the requirements are lower, but so is the efficiency.*

RECHARGE

Recharging the cell phone was less of a hassle. After I placed a cigarette lighter accessory socket on the battery carrier, I was able to plug the cell phone's car adapter charging cable directly into my portable storage source. It uses a simple zener diode as a regulated 5-V charging source for the cell phone. The draw on the portable battery is ~350 mA of initial charging current. The phone will fully recharge in about 5 h with a final current draw of ~50 mA, so the portable battery is required to supply ~9 Wh for a total cell phone charge. The solar array's output was ~3 W, which is about 3 h worth of sun for a total charge.

BENDING THE RULES

The issue of total portability is based on a delicately balanced system. If you can't gather enough energy from the sun (or another source), then eventually your portable power source will become depleted. If your portable storage system can't sustain usage over some nominal period of time (i.e., to take care of cloudy days), then you will also run out of electrons. If you happen to be located in an area outside of cell coverage, it doesn't matter how much power you've collected; you'll never be able to make that all-important connection.

During my three-week stint away from the office, I received close to 300 legitimate e-mails, above and beyond the normal amount of spam. I only lost the cell connection a total of three times over many hours of continued service. Although there was no actual cell time charge, the investment can't be overlooked. If you already have a web-capable digital cell phone and you disregard the cost of the phone itself, then the Internet connection costs for my phone were around $150 for the StarTAC interface cable and laptop software (Mototola's connectivity kit). There is an $8 monthly access fee, in addition to the monthly cell phone plan, for allowing you to make a connection to your ISP, via the cell phone system.

While camping, I tried keeping this Internet thing a secret by logging on in the late evenings (see Photo 1), after the boys had retired for the night. Some of the rules we have when we go camping are no radios, CD players, or Gameboys because we want the kids to experience the beauty of nature without outside distractions. However, one morning my son was asking some rather inquisitive questions. It seemed that in those nighttime hours he heard the familiar beep of Windows loading. The jig was up. He was content with my explanation of need and never mentioned it again. The adults, by comparison, were constantly badgering me to send e-mails to their wives. It just goes to show ya', sometimes it's tough to tell the men from the boys.

Jeff Bachiochi (pronounced "BAH-key-AH-key") is an electrical engineer on Circuit Cellar*'s engineering staff. His background includes product design and manufacturing. He may be reached at jeff.bachiochi@circuitcellar.com.*

SOURCES

StarTAC cell phones and accessories
Motorola, Inc.
(847) 576-5000
Fax: (847) 576-5372
http://www.gx-2.net/wwow/phones.html

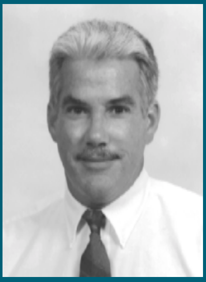
MAX1771
Maxim Integrated Products, Inc.
(408) 737-7600
Fax: (408) 737-7194
www.maxim-ic.com

Free ISP
freei
(253) 796-6500
Fax: (888) 841-9825
www.freeinternet.com

SILICON UPDATE

Tom Cantrell

Hot Chips 12



Hot Chips is one of Tom's favorite conferences because nothing is more exciting than the latest and greatest. There's no question that the hottest new chips have potential, but just how practical are they?

Nowhere are the chips hotter than on the 'Net, judging by the action at this year's Hot Chips conference.

Long-time readers know that Hot Chips is one of my perennial summertime favorites. The conference's focus is bleeding edge, performance-at-any-price silicon, which is a fun change from the workaday chips I usually cover. But, remember that when it comes to silicon, today's bleeding edge is tomorrow's mainstream.

NET WORTH

So it is with this season's collection of haute network processors. The silicon wizards are banking on networking to fuel the next wave of demand, and I think they're right on the money.

I don't claim to have a better idea than the next stuttering dot-com of where this is all is heading, but it doesn't take a rocket scientist to understand that the 'Net will be a huge deal. The only way to find out what the 'Net is good for is to hook everything up and see what happens.

As you might imagine, hooking everything up will swallow silicon at a prodigious pace. How many chips does it take to rebuild, rewire, and retrofit the global communications infrastructure? It would take a lot, and they'd better be hot!

TO EVERY SEASON

As it was with signals, graphics, and multimedia before, the networking crowd is demanding its own flavor of network processor instead of having to make do with a lash-up of boring standard chips and messy ASIC add-ons.

The premise underlying special-purpose processors is that a performance advantage can be obtained over a general-purpose design. However, marginal improvement isn't enough to overcome standard chip inertia and economies of scale. The gain has to be significant (at least on the order of 2×)

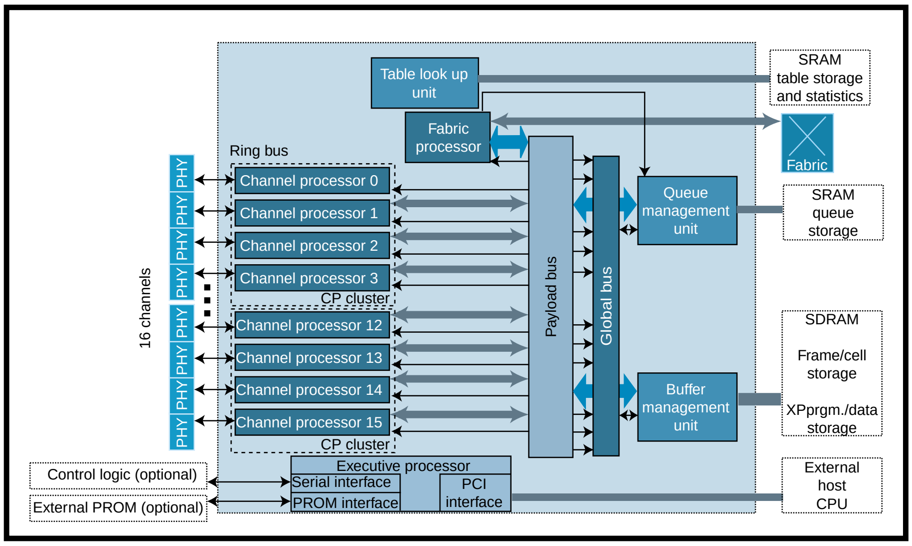


Figure 1—*The C-5 sets the stage for the next generation of computing with plenty of processors and a lot of memory on a single chip.*

Application	Data touch	State touch	Compute	Hybrid
Switching	Low	Low/Med	Low/Med	Yes
Routing	Low	Low/Med	Low/Med	Yes
QoS	Low/Med	Low/Med	Low/Med	Yes
Stateful firewall	Low/Med	Low/Med	Low/High	Yes
Proxy firewall	Med/High	Med	Med	Depends
Load balancing	Med	Med/High	Low/Med	Yes
CB load balance	High	Med/High	Low/Med	Yes
VPN	High	Med	High	*
Virus detection	High	High	High	No
IDS	High	High	High	No

* Crypto processing requires a special processor

Table 1—*Here are networking applications characterized by their data, state, and compute requirements. Many applications are hybrids where some packets involve a lot of processing to establish flow state, and subsequent packets are simply forwarded. The most demanding apps must process all traffic content in real time.*

before it becomes compelling. Whether or not that happens depends on the characteristics of the application (i.e., how amenable it is to optimization and acceleration by special-purpose features).

Because I'm certainly no expert on the inner workings of networking apps, the opening day tutorial, "Introduction to Network Processors," by Chuck Narad and Larry Huston of Intel, was a great way to kick off a conference that would see nearly a dozen new networking-related chips take a bow. [1]

Narad and Huston began by classifying particular networking tasks in terms of the degree to which the processor must touch, or further compute, data and state (see Table 1). Traditional routing and switching apps spend most of their time doing little more than forwarding data from A to B. On the other hand, new and anticipated features such as QoS (Quality of Service) and VPN (Virtual Private Networks) call for detailed inspection of the traffic and on-the-fly, data-driven decision making. In any case, with bandwidth aspirations of 1 Gbps and beyond, time is of the essence.

In answer to their own rhetorical question ("Why not just use a GHz+ Pentium?"), the instructors pointed out that locality of networking apps is poor. In what's essentially a dataflow problem, there's usually little relationship between a packet and other recently processed packets.

What that means is that the fancy cache schemes found on desktop chips not only don't help much, but cache thrash may make matters worse. Indeed, as wire speeds increase, a particular piece of netgear more likely aggregates traffic from disparate sources, exacerbating the problem.

Also filed under the heading of "Cruelty to Memory" is the fact that networking data structures rarely cooperate with the DRAMs that are required to buffer all the bits along their way. For instance, DRAMs work best for power-of-two burst transfer lengths, but the 'Net is littered with weird 53-byte ATM cells and Ethernet packets of variable lengths. There are also alignment problems as packets move up and down the stack, getting encapsulated and de-capsulated along the way. For instance, Ethernet headers are 14-bytes long, and a higher layer (such as PPP) may introduce its own header jitter of variable length.

In short, networking applications lend themselves to partitioning between what's referred to as control and data (i.e., forwarding) planes. Packets that require connection setup (and tear down) and other significant state modifications are handled by the control plane, and the data plane handles the bulk of the byte blasting.

As you'll see, various flavors of such partitioning are the primary differentiating factors that represent an opportunity for specialized processors to make their mark. Now, without further ado, on with the chips.

BIG BIT BANGERS

By my count, there were a total of nine networking chips presented at the conference, mostly from new startups jumping on the bandwagon. There's no hope of covering each in detail, but a few examples will serve to demonstrate the major trends.

As usual, the most obvious trend is that the hot chips are hotter than ever—bigger, faster, and more complicated. I must say, it's a challenge to keep my jaw from dropping when I see a chip like the C-5 DCP (Digital Communication Processor) from startup C-Port (recently acquired by big leaguer Motorola).

As you can see in Figure 1, C-5 starts with 16 individual channel processors. Each is a Harvard RISC with its own memory (24-KB instruction and 48-KB data), boosted with networking instructions and four register sets for efficient multitasking. Digging a bit deeper (see Figure 2), you can see that each channel processor incorporates dual (transmit and receive) serial data processors that are programmable enough to handle the many flavors of links that abound (10-/100-/1-Gb Ethernet, HDLC, ATM, SONET, etc.).

Then there's the executive processor that handles control plane stuff and host (PCI) interface. It functions with the aid of four specialized

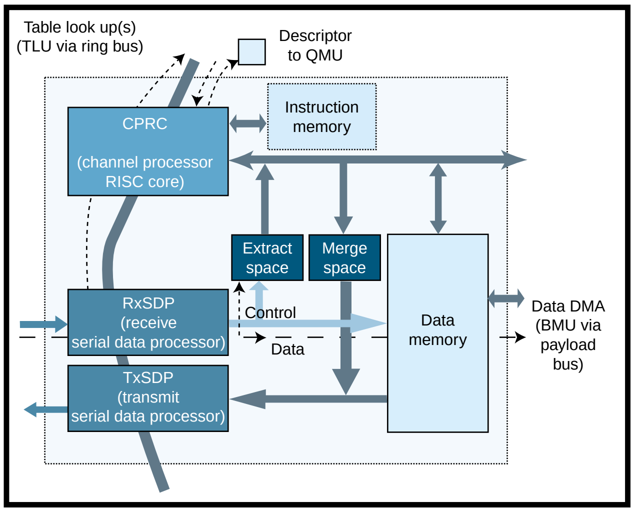


Table look up(s) (TLU via ring bus)
Descriptor to QMU
CPRC (channel processor RISC core)
Instruction memory
Extract space
Merge space
RxSDP (receive serial data processor)
Control
Data
Data memory
Data DMA (BMU via payload bus)
TxSDP (transmit serial data processor)

Figure 2—*Each channel processor in the C-5 has its own memory and additional serial data sub-processors that handle the ones and zeros.*

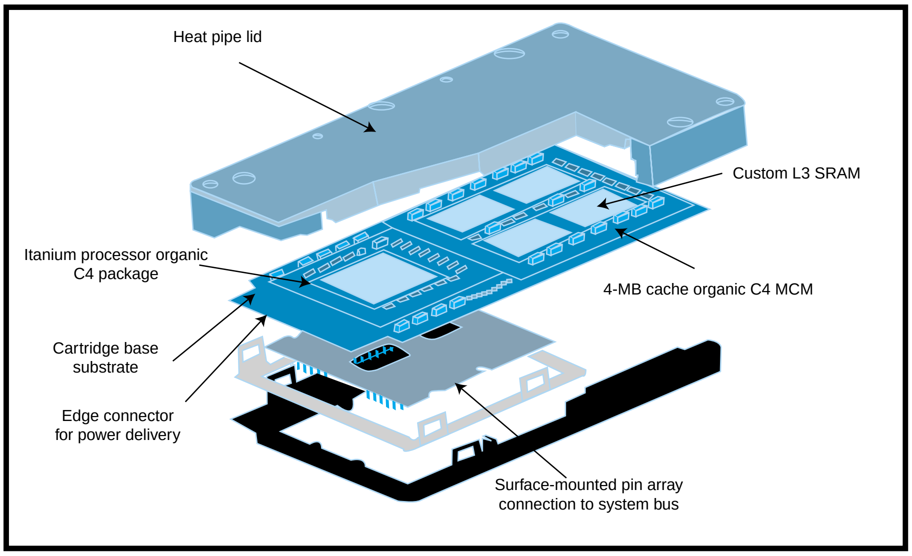

Heat pipe lid
Custom L3 SRAM
Itanium processor organic C4 package
4-MB cache organic C4 MCM
Cartridge base substrate
Edge connector for power delivery
Surface-mounted pin array connection to system bus

Figure 3—*As gains by architecture become more difficult to achieve, the Itanium cartridge indicates that implementation (cache integration, bus bandwidth, power management, etc.) will move to the fore.*

coprocessors for table look up, buffer and queue management, and 3.2-Gbps local (fabric) bus.

That's about 50 processors, over 1 MB of memory, and three internal buses with 50-Gbps bandwidth for a total of 56 million transistors, all tidily packaged in an 838-pin BGA!

The C-5 single-handedly illustrates a collective wisdom when it comes to networking chips. First, it's clear that the so-called chip multiprocessors (i.e., replicated processors on one piece of silicon) are moving from the lab to the market. Another example, the Vitesse NPU, incorporates four RISCs each, like the C-5, with four contexts so the chip can juggle 16 packets in flight at once.

Similarly, you can see that networking chips (and others) are starting to incorporate a ton of memory. Consider the iFlow address processor from Silicon Access Networks, bulked up with a whopping 52 Mb of DRAM!

Finally (although not limited to networking chips, but highlighted by the frenzy in that market), is the observation that the fab-less chip company model is key for innovation. Many of the bleeding edge networking chips announced were made by foundries, with TSMC (Taiwan Semiconductor Manufacturing Co.) credited for more than a few. The bottom line is that none of these hopeful startups would ever have a chance of raising enough money to make chips this hot themselves.

PROCESSOR PROGRESSION

Although most of the action was on the networking front, there were more than a few interesting presentations about traditional processors and DSPs as well. Certainly, everyone with a PC paid close attention to the Itanium presentations from Intel.

I don't need one yet, but Itanium looks like the trailblazer on the frontier of computing know-how. Most of the obvious concepts (pipelining, superscalar, vector, VLIW, cache, big register sets) were discovered long ago. For some time, CPU designers have faced a challenge going beyond the basics, and gains have come mainly from clock rate, bus bandwidth, and more cache, not from architecture. Worse yet, does Itanium still have to run clunky DOS games and all the other 'x86 detritus gathering dust on old hard drives? That's a challenge on the order of a moon shot.

The last time I wrote about Itanium, I covered the basic concepts such as EPIC (VLIW) and predication (conditional execution) and decided that, being at the end of my architectural rope and all, Intel had probably done as good a job as could be done.

It's a simple (but showstopping) dilemma; gains achieved by increasing circuit complexity are lost by the re-

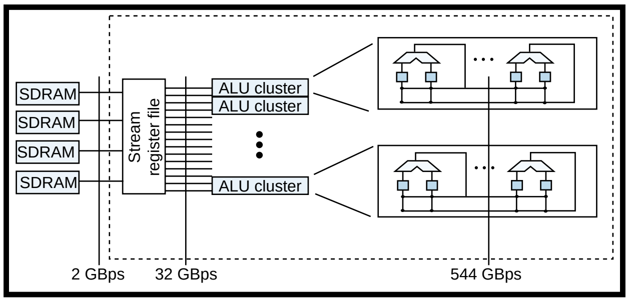


Figure 4—*Taking multiprocessing to the "exStream" calls for a lot of "IMAGINE-ation," and a lot of bandwidth.*

duced clock rate that results. There's indication that Itanium is at the edge, with architectural gains just sufficient enough to offset the faster clock advantage of a simpler design. Nevertheless, it's no secret that the term "Itanic" is heard in certain circles.

With the core architecture committed, implementation will decide the fate. The Itanium cartridge (see Figure 3) is a clear reminder that Intel really is, and has been for some time, a system rather than a chip company.

The cartridge is also a reminder of why Hot Chips is relevant, with the liquid-filled heat pipe recalling a Hot Chips presentation years ago by the Alpha crew. What's hot today will be on your desk tomorrow.

SHADES OF CRAY

The presence of (not to mention the rivalry between) two world-class universities has been key to the Silicon Valley phenomenon. Whether on the football field or in the computer lab, it's a big deal when the Bears (UC Berkeley) and the Cardinals (Stanford University) meet.

Between the two schools, they've set the course for computer research with notables like John Hennessy (Stanford) and David Patterson (UCB) at the helm, the pair responsible for tomes of near biblical stature (*Computer Architecture: A Quantitative Approach*). [2]

Both the Bears and the Cards were expected at the conference, so I was looking forward to seeing the latest big chips on campus. And thankfully, I was not disappointed.

Pushing the chip multiprocessor envelope, the Stanford team has aptly named its IMAGINE chip, foreseeing a future with not just a few, but hundreds or even thousands of processors onboard. Integrating 48 ALUs (8 clusters × 6 ALUs), it's especially well-suited for media and signal processing applications that have little data reuse (don't want or need cache), are highly data parallel (outputs independent of each other), and computationally intensive (60 ALU operations per memory reference).

With conceptually unlimited processing power on tap, the challenge for such designs is interconnect. IMAGINE uses a three-stage hierarchy comprised of external synchronous DRAM (SDRAM), global register files, and local (to each ALU) register files (see Figure 4).

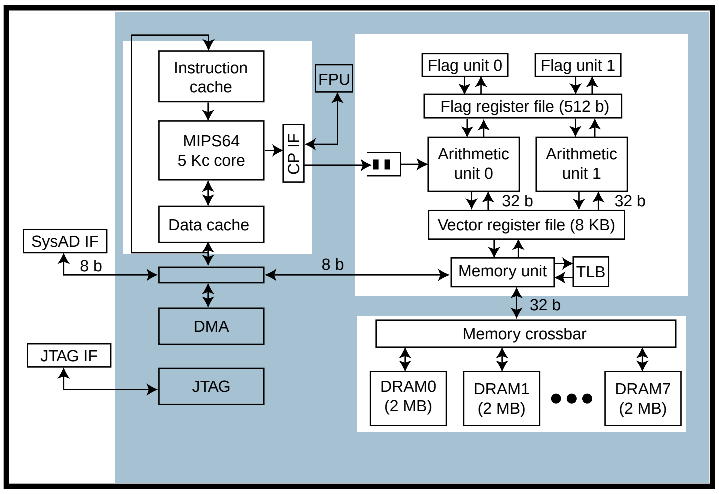


Figure 5—*Vector supercomputers aren't new, and neither is DRAM. But, put 'em together on a single chip and you've got something special—a VectorIRAM.*

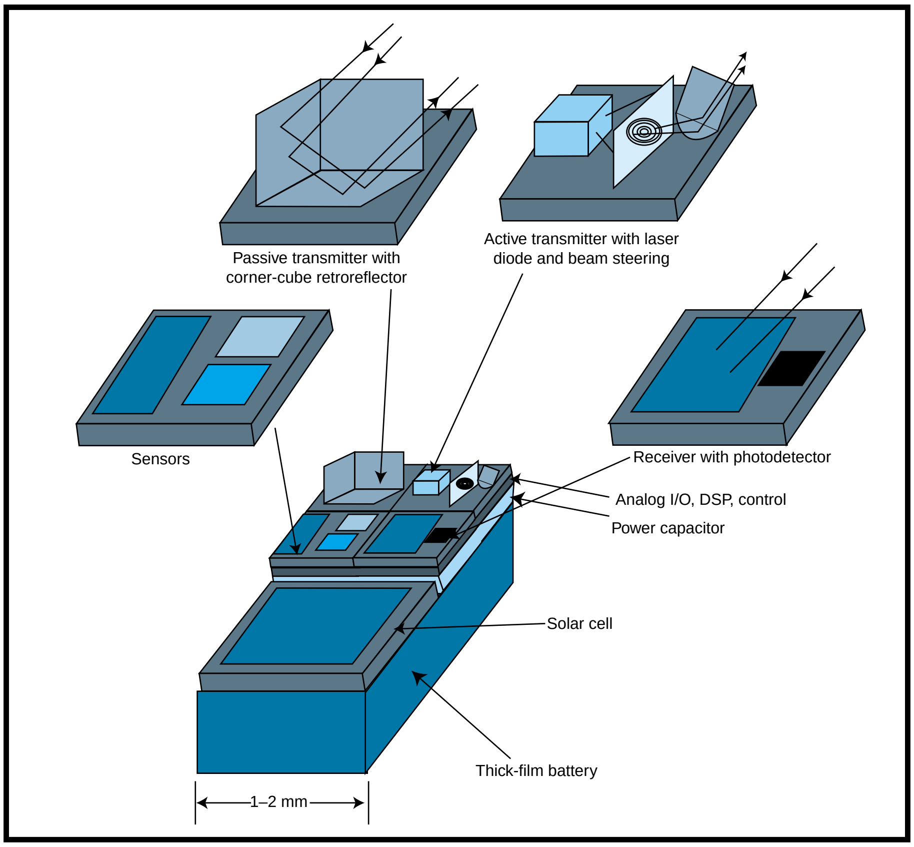

Passive transmitter with corner-cube retroreflector
Active transmitter with laser diode and beam steering
Sensors
Receiver with photodetector
Analog I/O, DSP, control
Power capacitor
Solar cell
Thick-film battery
1–2 mm

Figure 6—*If smart dust becomes a reality, I suppose we'll all be using Dustbusters instead of emulators.*

Working with TI and Intel, the IMAGINE team expects to take the 22-million transistor, 500-MHz, 10-GOPS bun out of the oven shortly.

By contrast, the Bears are pulling the memory integration strategy out of their playbook. Their Vector IRAM is what it seems to be: a vector processor with 128 Mb of DRAM thrown in (see Figure 5)!

The MIPS+ vector coprocessor part of the chip is pretty much the equivalent of a traditional vector supercomputer, a Cray-on-a-chip if you will. In fact, the IRAM compiler with C, C++, and Fortran frontends is based on well-regarded, time-proven Cray vectorizing compiler technology.

Because neither the vector register length nor the datapath width is explictly defined by the architecture, processing power is arbitrarily scalable. Once again, that puts the burden on bandwidth, where the on-chip 7.5-ns (page hit), 256-bit wide DRAM and 12.8-GBps per direction (load/store) crossbar switch shine.

I don't know who's going to win the big game, but both teams are sure giving it the old college try, wouldn't you say? For grad students, that's not bad at all. Maybe they'll find some time in between to get out and hit the beach or a party or two, but I doubt it.

WORLD BEYOND CHIPS

Whether it's the world's smallest PC, phone, or web gadget, when it comes to electronic stuff, smaller and cheaper is beautiful. Can silicon continue to deliver the goods?

I recall a speech by Intel's chairman of the board, Gordon Moore, at a Hot Chips conference some years back. He weighed in on the subject of a possible silicon wall, the proverbial end of the line for silicon. In a remarkably self-effacing way (for someone who's de facto chairman of the Valley), he frankly admitted that he was proven wrong in predicting a wall so many times that he finally quit doing it.

Whether or not there is a wall and when it might be reached is significant. When the price and performance of electronic products quits improving, falling to the ranks of things like insurance and hamburgers, then the party will be over. The business will become boring, Silicon Valley will crash, and the entire economy will suffer, losing the free lunch electronics-driven productivity gains now thoroughly enjoyed.

Folks at IBM's Almaden Research Center aren't twiddling their thumbs waiting for the ax to fall. Their presentation, "Towards Quantum Computation: A 215-Hz, 5-qubit Quantum Processor," foresees a day when 1 bit equals 1 atom. [3]

I must say, the presentation was a little over my head, what with all the bulk spin resonance, collapsing superpositions, Cooper pairs with Josephson junctions, and the like. But any bit-head would sit up and take notice of a machine that could, for example, search a database a thousand times faster or, dig this, factor integers more than a billion times faster!

Will it be ashes to ashes, dust to dust for the wizards? Those guys who are working on the "Preliminary Smart Dust Mote" at UC Berkeley sure hope so. Their goal: tiny autonomous motes in 1 mm^3 that proliferate like dust bunnies. Currently, they've been experimenting with COTS (commercial off-the-shelf) prototypes and have developed a capacitively actuated reflector passive optical communication scheme (see Figure 6) that's achieved 118 bps over 150-m range at less than 1 nJ per bit.

I'm an optimist at heart. If and when the wall does loom, I have to believe the wizards will climb right over it. May your chips stay hot, but if they don't, meet me at the Hot Atoms, Hot Dust, or Hot something or other conference.

Tom Cantrell has been working on chip, board, and systems design and marketing for several years. You may reach him by e-mail at tom.cantrell@circuitcellar.com.

REFERENCES

[1] L. Huston and C. Narad, "Introduction to Network Processors," HOT Chips 12: A Symposium on High-Performance Chips at Stanford University, www.hotchips.org, August 2000.

[2] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[3] Almaden Research Center, IBM, "Towards Quantum Computation: A 215-Hz, 5-qubit Quantum Processor," HOT Chips 12: A Symposium on High-Performance Chips at Stanford University, www.hotchips.org, August 2000.

SOURCES

mC-5 DCP
C-Port
www.cportcorp.com

iFlow Address Processor
Silicon Access Networks
(408) 545-1100
Fax: (408) 577-1940
www.siliconaccess.com

Itanium
Intel Corp.
(408) 765-8080
Fax: (408) 765-1399
www.intel.com

CIRCUIT CELLAR Test Your EQ

Problem 1—What does this function do? Why would you write a function like this?

```
char func1 (char x, char y, char z)
{
    char tmp;

    tmp  = (x ^ y);
    tmp &= z;
    tmp ^= y;

    return tmp;
}
```

Problem 2—Given a 16-bit digital signal at sample rate Fs, upsample it to a new sample rate of N×Fs by inserting N-1 zeros between each original sample. Then lowpass filter the result with an ideal sinc function to interpolate the N-1 samples. Next, dither and quantize this signal to 8 bits using, say, simple rounding. Lowpass-filter this result with the ideal brickwall filter with a cutoff frequency of Fs/2 Hz. Finally, decimate this result back down to sample rate Fs by throwing away N-1 out of every N samples.

Quantizing a signal results in the same total noise power no matter what the sample rate is. If we assume the quantization noise is white, then the quantization noise power in any fixed bandwidth (say, 0 to Fs/2) will decrease as we increase the oversampling ratio N. This way we can reduce the 16-to-8-bit quantization noise to an arbitrarily low level by using a sufficiently high oversampling factor.

Right?

Problem 3—The original Ferris Wheel was built for the Columbian Exposition in Chicago in 1893. It was the engineering highlight of the exposition and one of the most pervasive, lasting influences of the 1893 fair. The Ferris Wheel was Chicago's answer to the Eiffel Tower, the landmark of the 1889 Paris exhibition. The wheel was created by Pittsburgh, Pennsylvania bridge builder George W. Ferris. Supported by two 140 foot steel towers, its 45-foot axle was the largest single piece of forged steel at the time in the world. The wheel itself had a diameter of 250 feet, a circumference of 825 feet, and the maximum height was 264 feet. It was powered by two 1000-horsepower reversible engines. It had 36 wooden cars that could each hold 60 people.

Assuming that just one car is packed with 60 people that have an avereage weight of 150 lbs, how much torque is required in the worst case to turn the wheel? How fast can the two 1000-hp engines turn it under these circumstances?

Problem 4—When was the IBM PC first available to the public? What was Steve Ciarcia doing at the time?

What's *your* EQ?—The answers and 4 additional questions and answers are posted at www.circuitcellar.com.

You may contact the quizmasters at eq@circuitcellar.com.

8 more EQ questions each month in Circuit Cellar Online see pg. 2

PRIORITY INTERRUPT

Continuing the Plan

This afternoon I had to choose among a few tasks. I could spend the afternoon dragging in firewood for the solarium wood stove. I could fire up the leaf blower and try to finish moving tons of wet leaves before they freeze tonight and remain like concrete until spring. Or finally, I could keep an already rescheduled appointment at the dentist. As you might guess, they all lacked a certain amount of appeal.

Fortunately, before I had to choose among all these evils (I don't know that I'd call any of them lesser), Rob reminded me that I owed him an editorial. In truth, I didn't really need an excuse. Any opportunity to discuss *Circuit Cellar* should always take precedent over the dentist or getting a hernia lugging wood.

Seriously though, December is a good time to look back and discuss what we've attempted and what we've accomplished over the year. There was a time when we were simply a technical print magazine. Today, I'd like to think we have become a whole lot more than that. The plan was always in place but it took the explosive expansion of the Internet to properly launch it. 1999 and 2000 have been banner years for *Circuit Cellar*. We've increased our circulation, increased our editorial content, and doubled the number of design contests. When many print magazines are imploding, consolidating, or restructuring, *Circuit Cellar* continues expanding. The good news is that 2001 looks like more of the same.

Because hardware applications are near and dear to my heart, design contests were a high priority this year. It didn't hurt that big semiconductor manufacturers were lining up at the door for them either. We started the year with Philips Design2K (8051-family), did a little parallel online tango with the Microchip PIC2000 contest (PIC-family), and are ending the year with the Zilog Driven to Design contest (Z180-family). In my mind, contests have very real side benefits for both entrants and readers. $100,000 worth of combined prize money in this year's contests is ample incentive for entrants to stay on their design toes and all the great application articles that come from contests are a delight for the readers.

If a $20,000 first prize looks like enough incentive these days, you'll be happy to know that there is still time to enter the Zilog Driven to Design contest. If, on the other hand, you have always wanted to enter one of our design contests but just haven't seen your favorite processor yet, wait a minute—it will eventually show up! Before you burn up my e-mail in-basket, let me tell you that we are planning a couple more contests in 2001. The first one will be an Atmel AVR design contest starting in February. It will concentrate on AVR processors, FPGA's, and Atmel's new FPSLIC components.

The Internet continues to be an integral ingredient. Among our Internet achievements is the continued success of *Circuit Cellar Online* and ASK US. I suppose we should really be calling it the *Circuit Cellar* e-zine or something so that people aren't confused and think it is the same as the print magazine you are reading. It's not. Unlike many web sites that simply post rehashed HTML print articles, *Circuit Cellar* produces a second 100% original-material online magazine that is posted and available for free at ChipCenter.com (soon to be renamed eChips.com). If you like what you read here, make sure you check out www.chipcenter.com/circuitcellar/main.htm for a second dose of *Circuit Cellar* each month.

Finally, it's important to remember that everything you view at *Circuit Cellar* begins as an idea—and we don't have all of them. Many of the editorial products and ideas you see come from the readers. Our future depends on keeping the ideas flowing. If you've got a great inspiration or just think that we need to have a little prodding now and then, tell me about it. If I've tried to emphasize one idea more than any other over the years, it's that *Circuit Cellar* is a community and it's the contributions from the community that keep it running correctly.

Steve

steve.ciarcia@circuitcellar.com