EMBEDDED PC MONTHLY SECTION

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

#90 JANUARY 1998

10th ANNIVERSARY SPECIAL

Ciarcia and Bachiochi
Look at Lightning

Heterogeneous
Statecharts

Embedded Development
Using Serial BDM

RF Telemetry and PCs

$3.95 U.S.
$4.95 Canada

A Decade of Embedded Applications

I hadn't intended to get involved. But when Steve and company passed around a proof of Issue 1 of *Circuit Cellar INK* for the engineers to look at, I made more red marks in it than everybody else on the staff put together. It was then that they decided I should have a hand in creating future issues. That was ten years ago, and I've pored over every issue since.

We've seen a lot of changes in ten years, but I feel we've retained the same basic philosophy we started with. Give readers something they can take to the bench and apply to their next design. Tell them about your latest project, the newest whiz-bang chip on the market (but it must be available; no vaporware here), or a time-saving coding technique. Skip the advertorial found in any free trade magazine.

Certainly technology has changed, often making it more difficult for small business to afford the development tools. Integration has gone up, and more tools have migrated to the desktop. Software plays an ever-increasing role, and using off-the-shelf hardware can be more cost effective than rolling your own. These changes are reality, no matter how you might resist them.

Someone wondered during a recent Usenet thread whether the hardcore hardware experimenter has become a technodinosaur. I think if you still do designs other than those that are either trivial or educational using primarily discrete logic gates, you probably have become a technodinosaur. When an eight-pin processor is cheaper and uses less power and board space than a 555, two resistors, and a capacitor, it's foolish not to use the former to generate a waveform or do some kind of timing. By selecting an off-the-shelf component such as a processor to replace a handful of discrete parts, it quickly becomes a software problem instead of hardware.

Rather than software replacing hardware, though, today's computer engineer must be well versed in both hardware and software to most efficiently implement a design. It is this mixture that we've tried to reflect in our editorial coverage. No, we haven't forgotten our roots. Our roots have brought us to this point.

That said, I think it most appropriate that our tenth-anniversary issue deals with embedded development. And to kick it off, Steve and Jeff team up to update us on protecting against one of Steve's biggest nemeses: lightning. Gordon Doughman and Jim Sibigtroth show how processor designers are making things easier for programmers by building simple yet powerful debugging facilities right into the processor. Doron Drusinsky-Yoresh takes state diagrams into the next dimension with statecharts to better model complex real-time control problems. Finally, Avi Cohen looks at using not off-the-shelf hardware, but configurable off-the-shelf device-driver software.

In our columns, Tom Napier finishes his series on applying direct digital synthesis, Jeff moves to the front of the book with Steve, and Tom explores a bit of Motorola's reorganization by looking at their M•Core line.

In *EPC*, Scott Lehrbaum acknowledges the well-publicized year-2000 fiasco, but zeros in on how it may affect embedded designs. Raz Dan and Stefanie Hein ponder the make-versus-buy issue when it comes to embedded flash memory. Marc Guillemont kicks off *EPC*'s new Real-Time PC column by covering RTOS basics, and Fred begins a look at RF telemetry.

[illegible]

editor@circuitcellar.com

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

EDITORIAL DIRECTOR/PUBLISHER Steve Ciarcia	**ASSOCIATE PUBLISHER** Sue (Hodge) Skolnick
EDITOR-IN-CHIEF Ken Davidson	**CIRCULATION MANAGER** Rose Mansella
MANAGING EDITOR Janice Hughes	**BUSINESS MANAGER** Jeannette Walters
TECHNICAL EDITOR Elizabeth Laurençot	**ART DIRECTOR** KC Zienka
WEST COAST EDITOR Tom Cantrell	**ADVERTISING COORDINATOR** Valerie Luster
CONTRIBUTING EDITORS Rick Lehrbaum, Fred Eady	**ENGINEERING STAFF** Jeff Bachiochi
NEW PRODUCTS EDITOR Harv Weiner	**PRODUCTION STAFF** John Gorsky, James Soussounis

Cover photograph Ron Meadows – Meadows Marketing

Special thanks to Motorola for donating the MC68060 and MC68LC040 dies.

PRINTED IN THE UNITED STATES

HAJAR ASSOCIATES NATIONAL ADVERTISING REPRESENTATIVES

NORTHEAST & MID-ATLANTIC	**MIDWEST & SOUTHEAST**	**WEST COAST**
Barbara (Best) Curley	**Christa Collins**	**Barbara Jones & Shelley Rainey**
(561) 694-2044	(954) 966-3939	(714) 540-3554
Fax: (561) 694-2051	Fax: (954) 985-8457	Fax: (714) 540-7103
b.curley-Hajar@worldnet.att.net	HajarChrista@worldnet.att.net	shelley.hajar@worldnet.att.net

CONTACTING CIRCUIT CELLAR INK

SUBSCRIPTIONS:
INFORMATION: www.circuitcellar.com or subscribe@circuitcellar.com
TO SUBSCRIBE: (800) 269-6301 or via our editorial offices: (860) 875-2199
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 INK, 4 Park St., Vernon, CT 06066
AUTHOR CONTACT:
E-MAIL: Author addresses (when available) included at the end of each article.
ARTICLE FILES: ftp.circuitcellar.com

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

CIRCUIT CELLAR INK®, THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (860) 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, 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 INK Subscriptions, P.O. Box 698, Holmes, PA 19043-9613 or call (800) 269-6301.

Postmaster: Send address changes to Circuit Cellar INK, Circulation Dept., P.O. Box 698, Holmes, PA 19043-9613.

INSIDE ISSUE 90

www.circuitcellar.com

NEW PRODUCT NEWS

Edited by Harv Weiner

68HC05 DEVELOPMENT SYSTEM

The **HC05RTE** provides a low-cost means of debugging user code and evaluating target systems incorporating the Motorola J, P, K, and C series ’HC05 microcontrollers. The system, based on the MC68HC05C0 microcontroller, features host-computer downloading (via RS-232 communications port) directly to EEPROM or MCU RAM, a 4-KB monitor in 8-KB EPROM, an 8-KB EEPROM, and an inherent EEPROM programmer. A 50-pin connector and wire-wrap area is included for user interfacing, and a single 9–12-VDC supply is required.

The system includes an IBM-PC–compatible freeware cross-assembler, a flexible monitor program for debugging user code, 68HC05 manual, and 68HC05C0 datasheets.

Suggested retail price for the HC05RTE is **$168.05**. This system is also distributed by Active Components.

A.A.E. Systems
320 Yonge St., Ste. 116
Barrie, ON
Canada L4N 4C8
(705) 733-5611
Fax: (705) 733-1673
www.baradv.on.ca/aaesys

#501

OPTOISOLATION MODULE

The Telebyte **Model 269** is a compact RS-232–to–RS-232 optoisolation module that complies with the IEC 601-1 requirement that medical electrical equipment withstand 4000 VAC for 1 min. This level of isolation is provided for the RS-232 signals TD (Transmit Data), RD (Receive Data), and Ground. This capability is enhanced by the fact that the module does not require any external AC or DC power. All operating power is derived from the transmit data signals applied to each port.

The primary RS-232 port is a DB-25 male connector. In addition, the primary port may be configured, via switch selection, as a DTE or DCE device. The secondary port is configured in an RJ12 six-position modular jack with pins 3, 4, and 5 being Transmit Data, Receive Data, and Isolated Ground, respectively.

The Model 269 also guarantees compliance with the IEC specification for creepage and clearance. These terms refer to the absolute minimum spacing allowed between components and traces on the printed circuit board. In addition, each unit is 100% tested to verify the dielectric voltage withstand capability.

The Model 269 is packaged in a 2″ × 4.1″ × 0.75″ plastic case and sells for **$120**.

Telebyte Technology, Inc.
270 Pulaski Rd.
Greenlawn, NY 11740
(516) 423-3232
Fax: (516) 385-8184
sales@telebytetechnology.com
telebytetechnology.com

#502

NEW PRODUCT NEWS

DIGITAL STORAGE SCOPE

The **PC-MultiScope-2** from Mission Technology transforms any PC into a multitude of test instruments, including a full-function digital storage oscilloscope, frequency analyzer, digital voltmeter, adjustable DC power supply, strip chart recorder, and with an optional module, a function generator. The unit connects to a PC through a parallel port and features a Windows 3.1/95 graphical user interface.

The dual-channel unit features 1-MΩ input resistance, a 10-MHz analog bandwidth, triggering with AC or DC coupling, and an external trigger option built in. It provides programmable sampling intervals from 1 sample per 8 h to 20 megasamples per second, and a programmable gain from 10 V/div to 1 mV/div. It is auto-ranging in scope mode. Visual Basic source code is available separately.

The PC-MultiScope-2 comes complete with Windows-based software, an external AC power supply, parallel printer cable, and two alligator-clip 1:1 BNC probes. The unit sells for **$399**, and the **Visual Basic source code module** is priced at **$99**.

Amaze Electronics Corp.
4575 Grimsby Dr.
San Jose, CA 95130
(408) 748-7551
Fax: (408) 374-1737
amaze@hooked.net
www.hooked.net/users/amaze

#503

DISTRIBUTED MEASUREMENT DEVICE

The **KNM-DYN12 SmartLink** can make laboratory-grade measurements of force, acceleration, and dynamic pressure from remote locations. Its small size (6.7″ × 1.3″ × 1.1″) enables it to be located within inches of signals and sensors, minimizing lead-length errors and induced electrical noise. Measurements can be linked to a remote PC or controller, or the user can display and store results for local monitoring and debug via a digital readout device or palmtop PC.

The unit is equipped with an onboard microcomputer that provides data-acquisition, signal-processing, and communication capabilities. This setup enables the transfer of processed (rather than raw) data to a host computer.

Onboard memory means data can be collected and stored in the field for later analysis, eliminating the need to use a PC for data collection. Its ADC has a measurement resolution of 16 bits, and an analog recorder output, scaled from 0 to ±10 V, is provided.

Possible measurement configurations include eight two-wire analog inputs of pressure, force, and acceleration using low-impedance, voltage-mode piezoelectric sensors. Or, it can accommodate up to four inputs of very low frequency or static acceleration using capacitive sensors. One channel is available for a thermistor input, enabling temperature-compensated measurements.

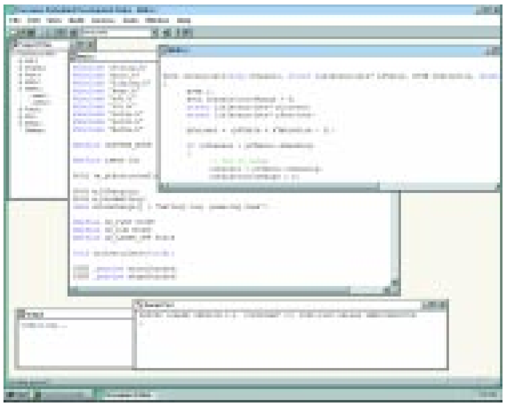
The KNM-DYN12 SmartLink sells for **$1880** in single quantities.

Keithley Instruments, Inc.
28775 Aurora Rd.
Cleveland, OH 44139-1891
(440) 248-0400
Fax: (440) 248-6168
www.keithley.com

#504

NEW PRODUCT NEWS

8051 C COMPILER



Crossware has released an **ANSI C compiler** for the 8051 family of microcontrollers that supports all variants of the chip. It also provides language extensions that give a C programmer full access to the underlying microcontroller architecture.

The compiler generates fast inline instructions with a minimum of library calls to produce high-performance embedded programs. When code size is an issue, the compiler uses a sophisticated merging process to combine repeated code sequences into separate subroutines.

Floating-point arithmetic is supported with 32- and 64-bit precision. With all floating-point routines hand-coded in 8051 assembler, complex floating-point algorithms can be calculated with speed and precision.

Its unique "smart pointer technology" enables pointers to work out for themselves which memory spaces they point to. The programmer is free to write ANSI-compatible code and avoid the inefficiencies of generic pointers.

The compiler is available for both DOS and Windows 95/NT 4.0. The Windows version comes complete with the Crossware Embedded Development Studio, which provides project management, build and browsing support, as well as context-colored drag-and-drop editing.

The DOS version costs **$1000**, and the Windows 95/NT 4.0 version sells for **$1300**. Exact U.S. prices depend on the exchange rate.

Crossware Products
St. John's Innovation Ctr., Cowley Rd.
Cambridge CB4 4WS UK
+44 (0) 1223 421263
Fax: +44 (0) 1223 421006
sales@crossware.com
www.crossware.com

#505

CIARCIA'S CIRCUIT CELLAR & FROM THE BENCH

Steve Ciarcia

Jeff Bachiochi

Ground Zero

A Real-World Look at Lightning

Living on a granite hill during a thunderstorm gives you a whole new respect for Mother Nature. To guard against paranoia, Jeff and Steve figure out how to automatically unhook their appliances before they become toast.

I know that some day I'll regret telling so many people this, but you see, we've had this little problem with thunderstorms. I know, you think I'm kidding. Isn't tornado alley in Oklahoma? Out west maybe, but Connecticut?

While storms are frequently predicted on hot summer afternoons around here, the reality is that there are very few severe storms and only one or two tornado warnings a year. That's the good news.

When a big thunderstorm occurs in Texas or Oklahoma, three vultures and a mountain goat might be the only ones who see the furious lightning display or even know it's happening. When it drops a 200-MPH tornado funnel, a few prairie dogs are often the only ones who have to rebuild their homes.

The bad news is that Connecticut is very small. Two traffic helicopters on opposite ends of the state have to be careful not to run into each other.

When a severe thunderstorm happens around here, everyone knows about it. When lightning strikes, it invariably hits something valuable. And, when a tornado funnel drops in a densely populated area, it doesn't miss.

Contrary to the paranoid description, however, my problem is not tornadoes. The infrequency and narrow path of a tornado make the odds of getting hit by one about the same as a 747 landing in the driveway.

My problem is lightning. And, location has everything to do with it. I live in one of the higher areas of Connecticut. By Colorado standards, it's barely a gopher mound, but 1000′ is high around here.

Unfortunately, underneath everything in this part of the state is probably the biggest slab of granite ledge east of the Mississippi. You can guess a few obvious consequences. For example, when my wife suggests I dig a hole, I don't even think about using a shovel. Short of dynamite, the only solution is the backhoe!

All that rock has an insidious consequence—earth grounding. More correctly, it's the lack of an effective earth ground that's the problem. Zap! Here's comes the lightning bolt, and where does it go? You guessed it—everywhere but down!

As you might expect, rock is a lousy electrical conductor. On my street, the earth-neutral grounding point at the electrical-service entrance is about as functional as the ground rod the electric company tries to drive into solid rock.

Prior to my all-out assault on the problem, I had a half-dozen lightning hits and over $20,000 in damages. My neighbor has had an electric blanket burned off his clothes line (it wasn't plugged in), seen flames coming from his power outlets, and had appliances blown off his counter. I've had TVs barbecued, computer equipment incinerated, and satellite systems destroyed.

The final straw was a few years back. While my wife and I were sitting on the front deck, we could hear a storm in the distance. Just as I said it would probably miss us, there was a brilliant flash and a deafening KAPOW!

A lightning bolt slammed into something right next to us. "Next to us" turned out to be my 15′ C-band satellite dish. I jumped up in time to see a cloud of steam rising from the recently melted LNA and what looked like smoke billowing from the garage. I grabbed a fire extinguisher and headed to the rescue.

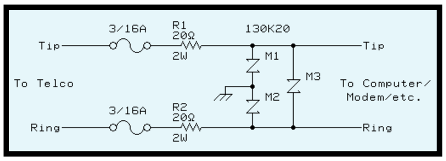


Figure 1—*The telephone line can serve as a lightning conduit. This circuit (shown in Photo 2) provides significant protection for both the phone equipment and user.*

Much to the EPA's chagrin, I'm sure, the smoke turned out to be freon. The lightning hit on the LNA had fed underground to the garage. Where the coax and control cables entered the house, the path of least resistance was out of the cable and into the central air conditioning lines.

As you might expect with all that power, it burned through the cables and copper tubing. Pow! Instant smog.

It was only because of the 2″ packed fiberglass insulation that I wasn't at a redwood-house–fueled wienie roast. When I saw how close we came to having everything torched, I got religion.

IT'S ALL IN THE GROUNDING

Scientists still don't fully understand lightning. Basically, it's a big discharge of static electricity that flashes toward the earth along a pilot leader.

This leader rushes down from the clouds in a series of discrete steps, ionizing the air as it goes. The final point is usually some elevated object on the ground. The bright lightning discharge we all see is the return stroke flowing back up the ionized path.

Any protection scheme doesn't prevent lightning from striking. It merely provides a low-resistance path for the lightning energy to ground. This path is the real issue.

A typical lightning bolt is 10–30 kA. The big strikes are as much as 100 kA (the power industry uses a 100-kA stroke with a rise time of 1.2 µs as its standard stroke).

Even if the path to ground were as low as 1 Ω, E = I × R tells us the DC voltage drop is 30 kV. If the resistance to ground is greater, then the voltage potentials are significantly higher.

Unfortunately, less technical discussions on the subject don't include the disastrous effects of inductance in this conduction path. Even with the massive lightning conductor used in the typical building lightning system, the inductance is on the order of 15 µH per foot.

The inductive voltage drop on a 20′ run of straight conductor with the industry stroke applied is on the order of one million volts! A conductor with lots of bends and twists has significantly higher inductance.

If the ground rod has a resistance of 10 Ω to ground, that adds another million volts along the path. Together, the total voltage floating around the building during the lightning strike is two million volts!

The voltage necessary to jump a spark through air is ~13 kV per inch, or 156 kV per foot. During a one or two million-volt strike on a building, you have to be careful about side flashes to any conductor that is grounded but not connected to the lightning system. That's why conducting bodies like equipment cabinets, machinery, metal rain gutters, and the AC electrical system have to be physically connected to the building's main grounding system.

When we casually speak of lightning taking the course of least resistance, we're talking about the flash-over. When lightning hits the cable TV line and isn't shunted to ground via a lightning arrestor and surge protector, the next stop is anybody's guess.

Short of putting up a tower to provide the proverbial zone of protection, defense comes by providing a conduction path with a lower overall impedance than alternative paths through your computer and fax machine.

The techniques are limited. The typical approach is to space lightning rods on the top of the building and use a heavy copper cable as a down conductor. Depending on the slope and area of the roof, there are standards regarding placement of the rods along the ridge versus around the perimeter (flat roofs are the most difficult).

If my house is any indication (part of the roof is shown in Photo 1), overkill is the typical installation choice. Counting the outbuildings, I have more than 25 lightning rods. The stranded copper cable is about a half inch in diameter, and 100′ of it weighs 20 lbs.

The fact that this is far from an exact science was illustrated by the professional installer's response to my observations. I pointed out that I've seen systems that employ sharp pointed rods as well as those that use large spheres. I've also seen light-gauge wire used as much as the heavy cable. His explanation was fascinating.

Apparently, there are two schools of thought in the lightning business. Most follow the convention that lightning hits the ground at a particular point because of the charge built up in that area of the ground.

By using very sharp points, the charge density at the point becomes high enough to leak off this accumulated energy, and it never gets high enough to attract a leader stroke. Because this happens over a reasonable period of time and at relatively low current, it also reduces the need for heavy stranded cable.

The other school of thought suggests that fate can't be deterred. If you're going to get hit, so be it. Just provide a good path to ground, and you'll be all right.

Round spheres handle the high energy density of a direct hit, and the heavy wire channels the load to ground. OK, so why is he installing heavy copper conductor and pointed rods, I ask? Insurance!

The points still supposedly reduce the target potential, but the heavy cable is there just in case that concept doesn't work quite as well as planned. I laughed.

The rods and down conductors are only half the system. It's the total impedance to earth ground that determines the voltage drop.

The building ground should have a resistance from 20 to 50 Ω. For most applications, this level is achieved simply by driving an 8–10′ copper-plated steel rod into the soil. Of course, the more conductive the soil, the better the ground.

But, my installation was at the other end of the spectrum—nonconductive rock without a lot of deep soil. The only solution was to create an artificial ground plane by burying cable around the perimeter of every building, attaching 25′ radial cables and ground rods (wherever they could be driven) every so many feet, and connecting all the building loops as one large grounding system.

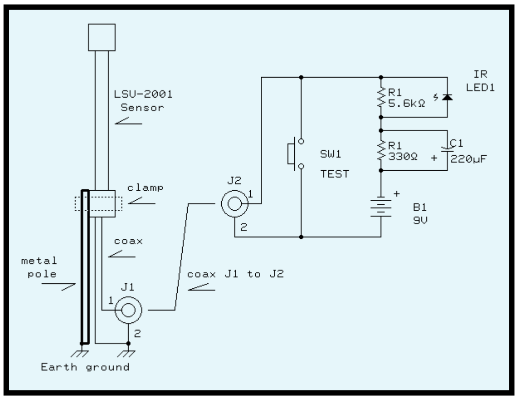


Figure 2—*An optically isolated pulse transmitter is connected to a low-cost McCallie Manufacturing lightning sensor mounted on a grounded pole on the roof.*

Jeff's a little luckier. His house has a steeply sloped roof and he only needs a few rods along the ridge. He also lives a hundred feet from a lake, so he also doesn't have the ledge or the grounding problems I have.

However, we both have a lot of sensitive electronics.

TRANSIENT VOLTAGE SUPPRESSION

Lightning protection falls into two broad categories—building protection and circuit protection. When lightning strikes, it creates an electromagnetic flux that radiates from the point of impact.

Like the windings of a transformer, this flux impulse induces a voltage on nearby conductors and electronic circuits. Depending on the proximity of the stroke, this transient voltage can be hundreds or even thousands of volts.

A number of techniques are available to protect electronic circuitry from the effects of voltage transients. These include the use of passive components (resistors and inductors) or devices with specific conduction characteristics to limit peak voltages.

The latter category includes gas discharge tubes (GDTs), reverse voltage breakdown diodes (TVSs), and zinc oxide varistors (MOVs). We'll just give you

#105

an overview for now, but next month, Joe DiBartolomeo starts a four-part MicroSeries with an in-depth look at surge suppression.

A GDT is a sealed tube containing an inert gas and two electrodes. When a high voltage appears across the terminals, the gas ionizes and a spark bridges the gap between the terminals, allowing current to flow.

The gas tube is like a crowbar device that short circuits the applied voltage down to less than 20 V. GDTs have very high surge capacity—on the order of 20 kA.

When a GDT is used across the AC power line, however, it must be combined with a circuit breaker. Once triggered by a transient, the GDT's 20-V clamping action effectively shorts out the 120 VAC as well.

The only way to reset the GDT is by blowing the breaker. GDTs are robust and efficient devices, but resetting the breaker from the crowbar action is a nuisance.

Typically, GDTs are used as main-power lightning arresters—often referred to as primary transient protection. Because their function involves a physical spark gap, they generally trigger at higher voltages than other protection devices. They can handle high current surges repeatedly without degradation.

Unfortunately, since their operation usually results in tripping the circuit breaker, GDTs are typically reserved for applications where a crowbar across the power line is a benefit rather than a nuisance.

Avalanche diodes are called by many different names (e.g., SAD, Transorb, TVS, etc). Basically, they're all just specialized zeners.

Their large PN junction blocks current flow until the voltage reaches a specific level when there is an avalanche of current flow. While considerably faster than GDTs, avalanche diodes are relatively low-current devices by comparison.

Their clamping characteristics are repeatable and do not degrade with continuous use (unless you exceed their surge-current rating). Avalanche diodes are ideal for low-voltage logic protection.

Photo 1—*Any effective lightning system starts with a good array of lightning rods spaced about every 20′ along the peak. The insert shows one rod in a little closer detail.*

While low-voltage avalanche diodes have some application as secondary surge protection, they are primarily used to protect semiconductor circuits from fast transients and ESD (electrostatic discharge). Avalanche diodes are frequently integrated within semiconductor components (e.g., communications line drivers) as well as attached across I/O lines and connecting wires.

Since they're designed to instantly clamp a transient and sacrifice themselves, avalanche diodes should be thought of as final protection.

MOVs are made of grains of zinc oxide bonded together in a disc form. They exhibit basic PN-junction zener characteristics. The typical 130-VRMS MOV (170 V_{peak}) has an initial conduction point of 205 V at 1 mA. Within 25 ns of reaching breakdown voltage, the internal resistance reduces from 5 MΩ to a level where as much as 10 kA can flow.

At this maximum current threshold, however, the MOV's clamping voltage can go as high as 600 V. This clamping threshold is determined by the MOV's grain density, and there is a wide range of operating voltages. MOVs are ideal for use both on the AC power line as well as low-voltage logic.

Because gas-tube crowbars and avalanche diodes need replacement when they sacrifice themselves, MOVs are used in 99% of power-protection products. Quite often, they're the only protective component in the device.

The only real downside to MOVs is that repeated high-current surges degrade their performance over time. In an application where a wide level of transient and surges are expected, GDTs are often used along with the MOVs.

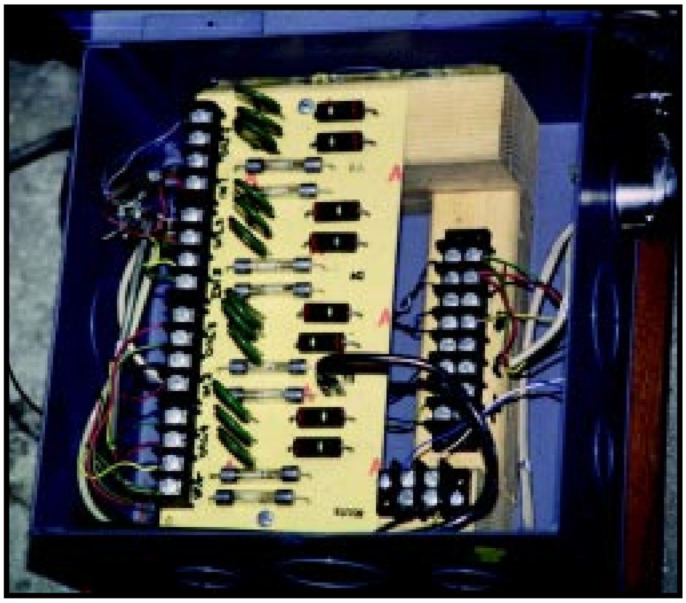


Photo 2—*The four phone lines entering the house have both differential and common-mode surge protection as outlined in Figure 1.*

MODEM AND POWER-LINE PROTECTION

Admittedly, I have a unique situation when it comes to lightning and surge protection. A simple lightning rod and single buried ground rod (like Jeff's) are more typical.

Once you solve the direct lightning threat, protecting a building from secondary invasions via the phone, cable, and power lines is the next order of business. There are two methodologies to this second line of defense.

The first method is to create a protective barrier using suppression devices. The second method is just a simple equipment-usage rule. When not in use, if the computer (or any piece of equipment) isn't plugged in, then nothing can hurt it. You have to decide which method is more practical for you.

All the discussion about various suppression components could lead you to believe we should use all of them. There are many exotic combinations of these devices, but those are generally intended for specialized applications.

Typically, a liberal sprinkling of MOVs provides a high level of protection at a reasonable price. The clamping voltage and physical placement of the MOVs are the only real issues.

There are two basic types of surges—common mode and differential mode. Common mode is when the surge potential is between the incoming line and the earth ground. A differential-mode surge is between two incoming lines with no reference to earth ground. All lines entering a building are susceptible to both.

Figure 1 illustrates a typical telephone/modem protection circuit. Photo 2 shows how I installed this circuit where my phone lines enter (the phone and cable companies provide the equivalent of a GDT connected externally).

Given the currents usually associated with phone communications, fusing might seem unnecessary. There isn't much that can help you in a close or direct lightning hit. If such an event occurs, the fuses are intended to simply disconnect the phone lines.

Interestingly, a telephone line is an isolated signal. Devices attached to it only require differential-mode protection. The two common-mode connected MOVs are there not to protect the modem or phone, but to protect the user.

While most phones use high dielectric plastic, a 10-kV common-mode surge could easily make the user be the path of least resistance to ground. The two common-mode connected MOVs prevent this.

Protecting the AC power line uses the same three-MOV configuration. Large MOVs, affectionately called doorknobs, are used at the power-line entry. Smaller MOVs (e.g., the 130K20) are used in the individual circuits or directly at the equipment power source.

UNPLUG THE COMPUTER!

The absolute best way to protect equipment from power-line, phone, and cable surges is to simply disconnect it when not in use. This seems obvious, but it's a nuisance having to plug and unplug entertainment centers and computers. It's only after you've had major damage that such inconvenience seems like a viable alternative.

Jeff is aware of my situation. I suspect that the reason he has lightning rods installed at all is from hearing my horror stories.

Unfortunately, while the concept is sound, implementation isn't that simple. Jeff has a big family and just can't unplug everything when he leaves for the office.

I have the luxury of unplugging the simple stuff like TVs and stereos, but devices like time-lapse recorders, auto-answering computers, and fax machines can't just be left unplugged. The optimum situation would be to unplug the equipment automatically only during dangerous conditions. When the storm passes, connections are restored.

We're aware that commercial devices exist to do this task. Unfortunately, their lofty price leaves them in the category of airport landing systems.

Without a source for a reasonably priced "thunderstorm switch," Jeff and I decided to make one. Conceivably, all it would take is a lightning sensor, decision logic, and a means to connect and disconnect the attached equipment.

AUTOMATIC THUNDERSTORM SWITCH

We can watch for rain, listen for thunder, and count the seconds after seeing the flash. These are the obvious indications of a threatening situation. There are many less obvious indicators as well.

The energy propagated from the current flow of a lightning strike contains wideband energy. Everything from 100 Hz to 100 MHz is produced.

Emissions below 100 kHz travel along the wave guide formed by the earth's surface and the lower ionosphere. With respect to the earth (ground), the air around the strike becomes charged, and there is a direct relationship between the amount of charge and the distance from the strike.

We located a minimum-cost lightning sensor from McCallie Manufac-

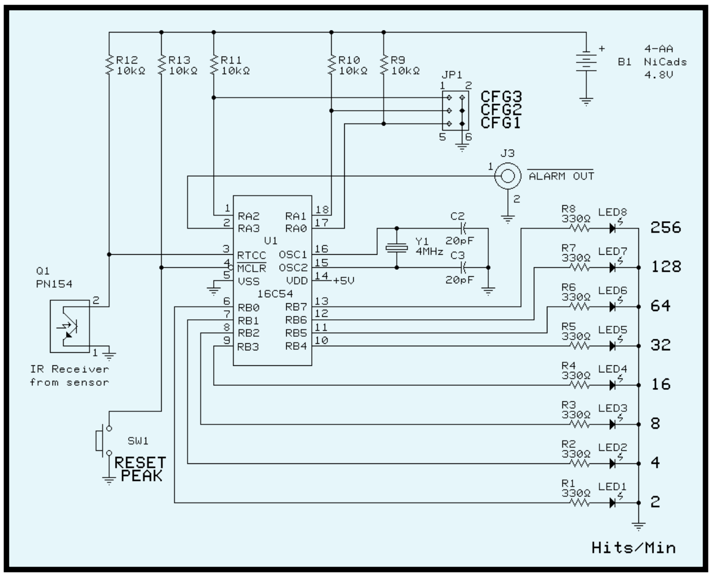


Figure 3—*The optical pulses are received from the lightning sensor and converted to a hits-per-minute LED indicator.*

Photo 3—*It will have to wait for next summer to be tested, but Jeff has attached the lightning sensor to the greenhouse roof.*

turing. Of the two models available, we chose the LSU2001, which is priced around $50.

Simple circuits are also provided for adding a meter or LEDs to monitor live data, or you can connect the sensor through an optocoupler to a PC. Optional software lets you count and graph storm data (providing you wish to keep the PC on day and night).

The manufacturer suggests mounting the LSU2001 on a well-grounded metal pole. The higher above ground it's mounted, the farther away you'll be able to detect lightning strikes.

Sensitivity is related to the differential charge between the air and ground—about 0.15 V/m. Put it twice as high, and it will be twice as sensitive.

They suggest that setting it 5′ high covers 15 miles, 10′ covers 50 miles, and 25′ covers 150 miles. The latter is enough to cover all of Connecticut, Rhode Island, and Massachusetts from our location.

Jeff wasn't enthusiastic about erecting a pole in his yard, and I wasn't volunteering to make like the Statue of Liberty.

However, since both of our roof peaks already had well-grounded lightning rods installed, attaching the sensor there made the perfect compromise location (see Photo 3).

The sensor has two wires leading out of its plastic enclosure. A coaxial connector would have made this a much cleaner job.

The coax was soldered to the wires and covered with tubing to make it watertight. The coax needs to be earth grounded so the sensor's internal circuitry can operate properly.

Interestingly, the sensor documentation comes with more warnings than any other piece of apparatus we've seen lately. Perhaps rightly so. If there's one thing you don't want, it's to provide a direct path for lightning into your house.

It's strongly recommended, for this reason, that the sensor's signals be isolated optically from your equipment and powered by its own battery. This setup also prevents line noise from interfering with the sensor. Figure 2 shows how it's done.

Battery longevity is essentially its shelf life. Power is only consumed when the air-to-ground potential rises above ~1.4 V. When this happens during a lightning strike, the circuit produces a pulse that flashes an infrared LED (LED1).

The IR LED points at a phototransistor directly or via a fiber-optic connection. The LED and phototransistor combination functions as an optoisolator. The greater the distance between them, the greater the isolation protection.

BLACK BOX IT

The sample data distributed with the sensor demonstrated that the effective number of hits per minute the sensor picked up during a thunderstorm ranged from 30 to well over 300 (within a 100-mile radius, I suppose this is acceptable).

Personally, I'd be heading for the cellar if I saw an indication of a 300 hits-per-minute storm, but Jeff was fascinated at knowing the actual quantity. He included eight LEDs to provide a visual display of the hits per minute as a power of two.

Using this method, the first LED indicates two hits; the second, four hits; the third, eight hits; and so on. The last LED indicates 256 or greater hits per minute. The LEDs are off under clear-sky conditions.

To count hits from the sensor, Jeff used the T0CK1 input on a PIC16C54 microcontroller. Figure 3 shows the circuitry for this simple display. Besides the eight directly driven LEDs and T0CK1, there are three bits for configuration and a single-bit alarm output. The configuration bits choose how the alarm output functions.

The output can be a 250-ms momentary pulse or continuously low during an alarm condition. The alarm trigger point is selected using the first two configuration bits. The four combinations select the hits-per-minute turn-on point of any of the upper four LEDs as its trigger level.

The software's main loop contains a 3-s counting period, followed by a total of the last 20 periods (total over the last 1 min.). The total counts are transposed into a byte with the proper bit high to enable an LED indicating the appropriate range.

Because you want to know if the storm is moving toward or away from your location, it's important to know about the past. Therefore, the `PEAK` count is displayed as a steady-state LED.

The `PRESENT` count is indicated by XORing the present count with the LEDs such that if `PEAK` and `PRESENT` are the same (as in a storm moving toward you), the LED flashes.

However, once `PRESENT` starts dropping, the peak-count LED remains steady and the present-count LED flashes. Now, you can tell immediately which direction the storm is heading. The `PEAK` value can be reset by pressing the reset button.

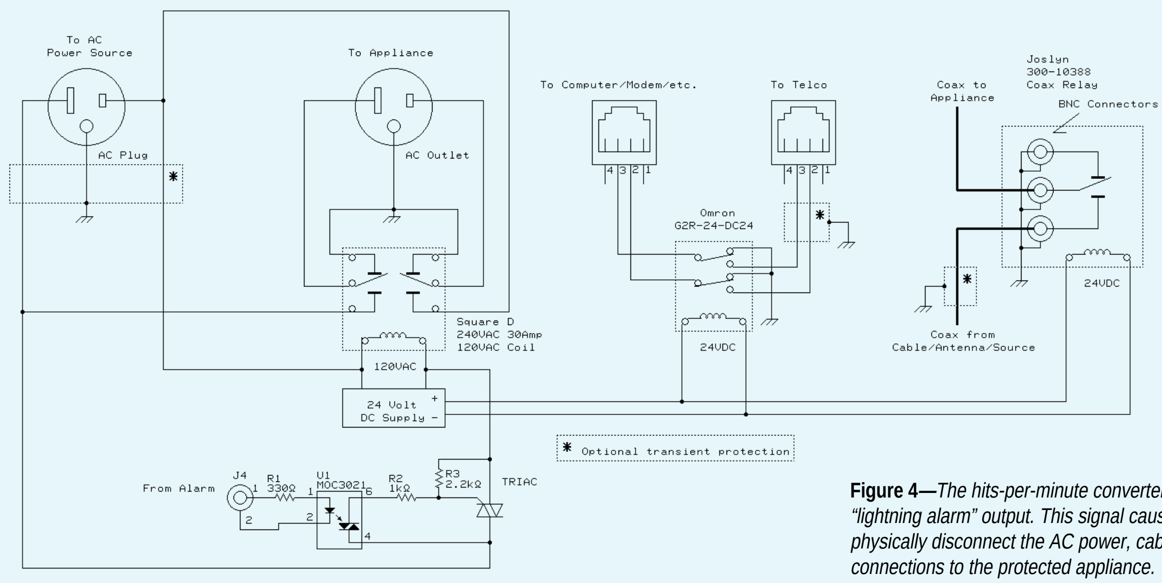


Figure 4—*The hits-per-minute converter also generates a "lightning alarm" output. This signal causes the circuit to physically disconnect the AC power, cable, and phone-line connections to the protected appliance.*

Although Jeff's software only takes up about a quarter of the 16C54's available code space, all of its registers are used. The majority are taken up by the 20 table entries doing the 3-s counting samples.

Only five other registers are used by the code for the rest of the functions. When no LEDs are on, the circuit requires only about 3 mA (add about 10 mA per LED).

Obviously, this whole circuit could operate from three alkaline batteries, but if it's mounted where you plan to view the thunderstorm's progress, trickle charging four NiCd batteries would be better. After all, when you need the information most to either pull the plug or tell you that conditions are all clear, you don't want to depend on a tired set of batteries.

Figure 4 is the switch's connect/disconnect section. This particular configuration accommodates AC power, coax, and phone lines.

The 16C54's alarm output (set for steady-state output mode) drives an optoisolated triac switch controlling the AC power relay. The circuit's normal condition is for the alarm output to be high and the relays energized. A small DC power supply, connected in parallel with the AC power relay coil, controls the coax and phone relays.

The switches' output connections are made to the normally closed contacts. When the power is off or an alarm condition exists, the relays are deenergized. This connects the equipment side of things to ground, where it provides the greatest protection.

While the normal spacing of the relay contacts is less than ¼″, should a high-voltage surge enter the line side of the switching unit, any place it arcs within the relay will be at ground. Certainly, if you incorporate MOV suppression in addition, such voltage levels shouldn't even exit at the relays.

We didn't include those MOVs and surge-suppression components on the schematic, but we indicated their proper placement. If you're already using protection devices on these lines, you may not need to include them inside as well.

WAITING FOR SUMMER

Winter isn't the best time to test lightning-detection equipment in Connecticut. We'll have to wait a few more months before the circuits get a real workout. Jeff might have his set to trigger on the 30 or more hits per minute, but I suspect I'll want to start thinking of alternatives at 2!

Considering how many of you live in southern states (or the Southern Hemisphere, for that matter), we're sure a number of you will have the opportunity to thoroughly test all this under ideal conditions long before we will.

We invite you to let us know about your tales of discovery and what we can expect. Perhaps by next fall, we'll have sufficient information for an update to the design. ◼

Steve Ciarcia is an electronics engineer and computer consultant with experience in process control, digital design, and product development. You may reach him at steve.ciarcia@circuitcellar.com.

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

SOURCES

Stormwise LSU-2001 lightning sensor
McCallie Mfg. Corp.
P.O. Box 8631
Greenville, TX 75404-8631
(903) 383-7047

PIC16C54
Microchip Technology, Inc.
2355 W. Chandler Blvd.
Chandler, AZ 85224-6199
(602) 786-7200
Fax: (602) 786-7277
www.microchip.com

I R S

401 Very Useful
402 Moderately Useful
403 Not Useful

FEATURE ARTICLE

Gordon Doughman & Jim Sibigtroth

’HC12 Development Using a Serial BDM

As the speed and complexity of embedded micros increase, it’s more difficult to build high-speed in-circuit emulators. As a result, many offer on-chip BDM interfaces. Gordon and Jim show how to debug using these resources.

The MC68HC912B32 is one of the newest members of the MC68HC12 family of 16-bit microcontrollers that combines flash memory and byte-erasable EEPROM on the same chip. In addition to the considerable number of on-chip peripherals, the ’HC912 contains two powerful on-chip modules that enable nonintrusive debugging of the user’s target system.

The Background Debug Module (BDM) consists of a single-wire hardware interface that provides nonintrusive access to the target’s memory, control of the target CPU execution, and access to the CPU’s registers.

The Breakpoint Module provides one or two hardware breakpoints that enable easy debugging of software contained in either on- or off-chip nonvolatile memory.

In this article, we examine the capabilities of these two on-chip debugging aids and the development-tool support provided by the ’HC912 evaluation board.

THE ’HC12 BDM

As the speed and complexity of embedded microcontrollers increase, it is more difficult to build the high-speed in-circuit emulators required to design and debug embedded systems. This is especially true for systems built around microcontrollers that have large amounts of on-chip nonvolatile memory and are operated in single-chip mode in the target system.

When attempting to design such an emulator, the designer typically places the single-chip microcontroller in expanded mode and replaces the on-chip nonvolatile memory with off-chip RAM. In addition, a specially designed ASIC or FPGA must replace the port pins lost to the address and data buses because the part is operated in expanded mode.

These emulators provide sophisticated debugging facilities like multiple hardware breakpoints with complex triggering capability and built-in bus state analyzers. However, their high-speed components make them quite expensive.

To help combat the increasing cost and complexity of emulating sophisticated single-chip microcontrollers, Motorola originally introduced a BDM interface on its M68HC16 and M68300 families of 16- and 32-bit microcontrollers. A BDM system enables simple, low-cost debugger hardware to connect to a target system through a dedicated high-speed serial interface.

While the BDM system greatly increases the capabilities of low-cost development tools for these product families, it has one major drawback. Because these systems are part of the CPU cores, the BDM can’t be used unless the CPU halts execution of the target-system software.

The ’HC12 BDM module reduces the physical interface to a single pin (BKGD), so the target system can be accessed with two connections—a common ground and the single background interface pin.

Also, because the ’HC12 BDM module isn’t part of the CPU12 core, it allows reading and writing of target memory without disturbing a running application.

Other BDM commands (e.g., examining and/or changing CPU registers and executing single instructions) require that the application program be stopped and the BDM system be placed in active mode, but the background-debug system is usable in any operating mode.

While the minimum required physical interface to the 'HC12 BDM is the BKGD pin and ground, Motorola defined a six-pin BDM header for the 'HC12 family that includes several other connections. The header, shown in Figure 1, also contains connections to the target system's V_{DD}, MCU reset, and V_{PP}.

The V_{DD} connection enables the target to be powered by the BDM debug pod or vice versa.

Connecting the MCU reset pin to the BDM pod lets the pod gain control of the target system by asserting the MCU reset line. This capability is useful if target-system software becomes stuck in a loop and won't respond to any target-system stimulus.

In addition, resetting the target system in special single-chip mode (by holding the BKGD, MODA, and MODB pins low during reset) places the target system in active background mode. This situation lets the BDM pod gain control of any 'HC12-based system before software is placed in the target memory.

The V_{PP} pin on the BDM header can optionally connect to the Vfp pin of the target 'HC12 MCU. This connection means the flash programming voltage can be supplied by the BDM pod rather than duplicating the Vfp-generating circuitry on each target board. It also allows a single-point connection during the debug/reprogram cycle.

For a detailed description of the 'HC12's BDM, including a complete explanation of its custom serial communication protocol, see Jim's article, "A Single-Wire Development Interface" (*INK* 72).

THE 'HC12 BREAKPOINT MODULE

The hardware breakpoint module in the 'HC912 is a large 34-bit comparator that can be configured to operate in one of three modes.

In BDM full-address/data mode, the comparators look for a match on the 16-bit address, 16-bit data, and R/W signal. Address low byte, data high byte, data low byte, and R/W can be separately configured for don't care.

When a match is detected, an interrupt causes the MCU to enter active background mode at the next instruction boundary. A typical trigger condition using this mode is "trigger when a specific data value is read from the address of the SCI receive data register."

In BDM dual-address mode, the comparator is split into two separate 17-bit sections to allow triggering on either of two user-specified addresses. The BKPM control bit lets these breakpoints use either an interrupt or a tagging mechanism to respond to trigger matches. In this mode, the address low byte and/or the R/W signal can be set for don't care. (R/W is ignored when the tagging mechanism is used.)

The interrupt mechanism causes the MCU to go to active background mode at the next instruction boundary after a match. The tagging mechanism produces a tag when the address matches.

This tag follows along in the instruction pipe until the tagged instruction would execute. This mechanism causes the breakpoint to enter active background mode just before the instruction at the selected breakpoint address where it would have executed. This mode corresponds most closely to what traditional debuggers call breakpoints.

In the third mode—SWI dual address—the comparator is split into two separate 16-bit sections to allow two program address breakpoints. Each address low byte can optionally be set for don't care to allow triggering within a 256-byte area of memory.

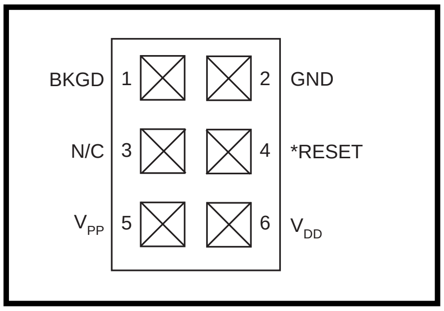


Figure 1—*Motorola defines a standard six-pin (2 × 3) 0.025″ square post header as a target-system BDM connector.*

This mode uses the tagging mechanism to cause an `SWI` instruction to be executed instead of the instructions at the two selected addresses. Given some simple planning, you can use this mode to perform code patches in flash memory or ROM.

The SWI vector needs to use an indirect pointer in RAM or EEPROM so the service routine can be added at a later date without erasing the flash or changing the masked ROM.

D-Bug12 uses the BDM dual-address mode for breakpoints after `USEHBR` executes. Versions earlier than V.2.0.2 didn't support hardware breakpoints, but anyone with an EVB912B32 evaluation board can get a simple S-record upgrade file from the Motorola or Circuit Cellar Web sites.

EVALUATION BOARD

The EVB912B32 is a small, economical evaluation and debugging tool that uses the specialized debugging modules on the 'HC912 MCU.

As you see in Figure 2, the design is extremely simple. The board contains only three active components—the 'HC912 microcontroller, an RS-232 level translator, and an undervoltage-sensing circuit.

Access to the 'HC912's powerful debugging features comes from the firmware located in the HC912's on-chip flash memory and two 6-pin connectors—BDM IN and BDM OUT.

The D-Bug12 monitor program can operate in two different modes. In EVB mode (see Figure 3), the firmware operates as a flash-resident debugger. While this mode provides a stable environment for evaluating the 'HC12 architecture, testing new algorithms,

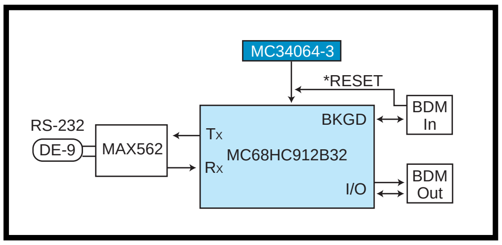


Figure 2—*The M68EVB912B32 evaluation board/debug pod has only three active ICs. A DE-9 connector allows connection to a host PC. BDM IN and BDM OUT are six-pin headers enabling the EVB to act as a target system or debug pod, respectively.*

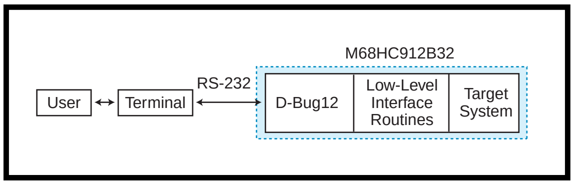


Figure 3—*In EVB mode, a terminal emulator in the host PC interfaces via RS-232 to the evaluation board. D-Bug12 (in the flash memory) interfaces with on-chip memory and peripherals through a set of low-level interface routines.*

or conducting performance benchmarks, it has some limitations.

Because D-Bug12 executes out of 'HC912's internal flash memory, the flash, half of the RAM, and the SCI serial port are not available to the developer. In EVB mode, D-Bug12 can't support true emulation of a target system.

When D-Bug12 operates in Pod mode, the low-level interface routines accessing the EVB912B32's on-chip resources are replaced by routines that implement the custom serial protocol of the 'HC12 BDM interface. These low-level software drivers communicate with the target through the BDM OUT connector, as shown in Figure 4, to nonintrusively access the 'HC12 target system.

D-Bug12 uses a simple command-line interface to accept user-entered commands and display requested data. When operating in EVB mode, D-Bug12 displays the single-character > prompt when waiting for a user command. In Pod mode, it displays one of two prompts depending on the target system's state.

When the target system is in active background mode (not running a user program), the two-character prompt S> is displayed to indicate that the target microcontroller is stopped. The two-character prompt R> indicates the target is running a user program.

Because the 'HC12 BDM interface allows reading and writing of target-system memory even when running a user program, D-Bug12 is always available for command entry when operating in Pod mode. Any D-Bug12 commands that alter or display target memory may be entered when either prompt is displayed. Table 1 lists the D-Bug12 commands and indicates when they may be used.

DEBUGGING IN FLASH

In addition to using the EVB912B32 with D-Bug12 as a simple evaluation or BDM debugging tool, the EVB912B32 may also be used as a target system. Using two EVBs, the BDM OUT of one can be connected to BDM IN of the second.

This setup enables one EVB to be used as a nonintrusive BDM debugger while the other functions as a target board operating in single-chip mode. Because D-Bug12 uses the hardware instruction tracing capability of the BDM module and supports the hardware breakpoint module, applications can be developed and debugged directly out of the 'HC912's 32-KB on-chip flash.

At powerup or reset, D-Bug12 supports ten software breakpoints in either mode. Software breakpoints are implemented by replacing the opcode of an instruction to be executed with either a Software Interrupt (SWI) instruction when operating in EVB mode or a Background (BGND) instruction when operating in Pod mode.

Using this method to implement breakpoints enables D-Bug12 to support 'HC12 family members that lack a hardware breakpoint module. When developing programs with such devices, target program storage memory would have to be replaced with RAM to use D-Bug12's breakpoint functions.

The D-Bug12 command USEHBR lets the program developer switch from using software breakpoints to using the hardware breakpoint module of the target microcontroller. In Pod mode, the D-Bug12 uses the target's hardware breakpoint module in BDM dual-address mode to substitute two program-only hardware breakpoints for the 10 software breakpoints.

Hardware breakpoints may also be used when operating in EVB mode. In this case, D-Bug12 operates the resident 'HC912's hardware breakpoint module in SWI dual-address mode to provide two program-only hardware breakpoints.

Using hardware breakpoints in EVB mode provides two advantages over using software breakpoints. Because the 'HC912 CPU can't perform single-step instruction tracing, executing a single instruction is performed using temporary breakpoints.

Using software breakpoints with this method works fine for programs located in the 512 bytes of RAM available for user programs. However, it can pose a potential problem for debugging programs located in the on-chip byte-erasable EEPROM.

Because D-Bug12's memory-access routines transparently (re)program the on-chip byte-erasable EEPROM for writes to the EEPROM address range, at least one EEPROM location (two for branch instructions) will be erased and programmed twice each time an instruction is traced.

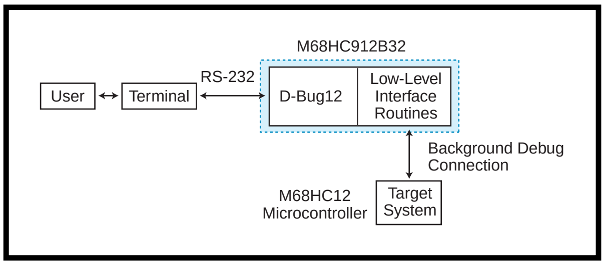
Repeatedly erasing and programming the EEPROM can lead to its eventual failure. Using the hardware breakpoints avoids this behavior when tracing instructions. Using hardware breakpoints in EVB mode also allows tracing through D-Bug12's user-accessible routines located in the on-chip flash memory [1].

SOLVING A REAL-WORLD PROBLEM

Although the 'HC912 BDM and hardware breakpoint modules are extremely useful during embedded-application development and debugging, they recently proved invaluable in helping track down a customer problem with the 'HC912's Byte Data Link Controller (BDLC) communications module.

The problem involved an occasional failure of the 'HC912's on-chip BDLC module that couldn't be traced to a particular event or series of events that occurred in the customer's system.

Figure 4—*In Pod mode, D-Bug12 communicates with the target system through an alternate set of low-level interface routines. The target system is now a separate 'HC12 system connected to the BDMOUT header.*



Tracking down the exact problem and finding a solution took several weeks, but the EVB912B32's support of the 'HC912's on-chip hardware breakpoint and BDM module helped us discover the underlying problem.

Because of the occasional and apparent random failure of the 'HC912 BDLC, there was initially very little data to work with. Since the problem had been isolated to a particular system that used the module containing the 'HC912, we didn't suspect the BDLC initialization or driver software. However, the customer's software was thoroughly reviewed and found to be correct.

The next step was observing the failure of the module in the customer's system while it ran under actual operating conditions. Because the site was 40 miles away from the engineering location, we didn't have the luxury of working with the software development engineer or the module's source code. We were armed only with an EVB912B32, oscilloscope, and J1850 bus-analysis tool.

Using the J1850 bus-analysis tool, we observed an unusually high number of errors during message transmission and reception. To get a better idea of the type and frequency of the received-message errors, we needed to observe the activity of the customer's code when servicing BDLC interrupts.

We decided to use the target system's hardware breakpoint module operating in SWI dual-address mode to patch into the customer's BDLC interrupt service routine. The patch routine would identify the type of the BDLC error interrupt, increment a 16-bit counter, and return to the customer's BDLC interrupt handler. Once the customer's target system was running, we used the D-Bug12 Memory Display (MD) command to nonintrusively examine the 16-bit counters.

In our first attempt to observe the software's actions, we attached the EVB912B32 to the BDM connector on the customer's module and executed D-Bug12's STOP command to place the target CPU in the active background mode.

This action immediately caused a target-system reset because the customer's software had enabled the COP watchdog-timer system. Resetting the module via RESET placed the target in

Command	Description	Use with S>?	Use with R>?
ASM	Single-line assembler/disassembler	Yes	Yes
BAUD	Set the SCI communications baud rate	Yes	Yes
BF	Block fill user memory with data	Yes	Yes
BR	Set/Display user breakpoints	Yes	No
BULK	Bulk erase target EEPROM	Yes	Yes
CALL	Execute a user subroutine	Yes	No
DEVICE	Select/define a new target MCU device	Yes	No
EEBASE	Define target's EEPROM base address	Yes	No
FBULK	Erase target's on-chip flash memory	Yes	No
FLOAD	Program target's on-chip flash memory	Yes	No
G	Begin execution of user program	Yes	No
GT	Set a temporary breakpoint and Go	Yes	No
HELP	Display D-Bug12 command summary	Yes	Yes
LOAD	Load user program in S-record format	Yes	Yes
MD	Display memory in hex/ASCII format	Yes	Yes
MDW	Display memory in hex/ASCII word format	Yes	Yes
MM	Interactively examine/change memory	Yes	Yes
MMW	Interactively examine/change memory words	Yes	Yes
MOVE	Move a block of memory	Yes	Yes
NOBR	Remove one/all user breakpoints	Yes	No
RD	Display target CPU registers	Yes	No
REGBASE	Define the target's I/O Register base address	Yes	No
RESET	Reset the target CPU	Yes	Yes
RM	Interactively examine/change CPU registers	Yes	No
STOP	Stop the execution of user code	No	Yes
T	Trace instruction execution	Yes	No
UPLOAD	Display memory contents in S-record format	Yes	Yes
USEHBR	Use EVB/target hardware breakpoints	Yes	No
VERF	Verify memory contents against S-records	Yes	Yes

Table 1—*All D-Bug12 commands are available from the S> prompt except STOP (target is already stopped). The last column shows which commands are available from the R> prompt when the target system is running.*

the active background mode, enabling us to gain control of the target CPU.

Using the D-Bug12 trace command, we analyzed the start-up code to determine where the watchdog timer, RAM, and other peripherals were initialized. Once these addresses were determined, the system was reset and two hardware breakpoints were set using D-Bug12's BR command.

The first breakpoint halted execution just before the watchdog timer was initialized. The second breakpoint stopped program execution just after the on-chip RAM was initialized.

This sequence let us skip over the watchdog-timer initialization sequence at the first breakpoint and then download the small BDLC interrupt service routine patch into an unused area of RAM after the second breakpoint.

As we mentioned, some advanced planning is required to use the Breakpoint Module in SWI mode to patch code in flash memory or ROM. Fortunately, the interrupt service routines in the customer's software were accessed through a jump table located in the on-chip RAM. When the software patch was downloaded into the unused area of the on-chip RAM, it also modified the SWI table entry to point to our code patch.

Finally, after continuing execution of the customer's software, we used D-Bug12's memory-modify command to initialize the hardware breakpoint address and control registers to use the SWI dual-address mode.

As soon as the control register was written, enabling the hardware breakpoint, we could examine the 16-bit counters being incremented by our code patch each time the BDLC received an invalid message. This information gave us the details we needed to determine the cause of the problem and was extremely helpful for finding a software workaround.

FINISH THE JOB

Working on this solution, one thing became very clear to us. As embedded systems become more complex, it's increasingly important for microcontroller manufacturers to provide advanced, nonintrusive debugging hardware as part of their microcontroller designs.

The nonintrusive BDM found on the 'HC12 family and the Breakpoint Module on the 'HC912 proved invaluable in solving a particularly difficult customer problem.

Gordon Doughman is a senior field applications engineer working on automotive and industrial systems software for Motorola Semiconductor. He wrote D-Bug12 and BASIC11, a BASIC interpreter for the M68HC11, as well as authoring several application notes for '68HC11 and 'HC12 microcontroller families. You may reach Gordon at RTNR30@email.mot.com.

Jim Sibigtroth is a system design engineer working on advanced microcontrollers for Motorola. He defined the CPU12 instruction set and several of the 'HC12 systems including the background debug system and memory expansion system. You can reach Jim at RFTP70@email.mot.com.

SOFTWARE

The S-record upgrade file for the EVB912B32 can be found on the Circuit Cellar Web site and at www.mcu.motsps.com/freeweb/mcu12_ndx.html.

REFERENCE

[1] G. Doughman, *Using the Callable Routines in D-Bug12*, App. Note AN1280A, Motorola, Austin, TX, 1997.

SOURCE

MC68HC912B32TS/D, EVB912-B32, D-Bug12, CPU12RM/AD, AN1280A/D
Motorola
MCU Information Line
P.O. Box 13026
Austin, TX 78711-3026
(512) 328-2268
Fax: (800) 765-9753
www.mcu.motsps.com

I R S

404 Very Useful
405 Moderately Useful
406 Not Useful

FEATURE ARTICLE

Heterogeneous State Machine Control

Doron Drusinsky-Yoresh

Automotive electronics have taken on a life of their own. Their complex interprocessing requires engineers to mix classic control with continuous modeling tasks. Doron shows how statecharts ease the difficulties.

Clearly, automotive electronics have gone through a major evolution within the last decade. Processors now perform dozens of tasks that are completely new or were previously managed using analog or mechanical technologies.

The well-known advantages of digital control and computation include flexibility, diversity, low cost, and rapid development. With the proliferation of digital processors in vehicles, new challenges are emerging, like management, synchronization, and scheduling of heterogeneous computational tasks.

In this article, I describe how complex state-machine control tasks, classical control tasks, and continuous modeling tasks combine to achieve the heterogeneous solution necessary in practical automotive applications.

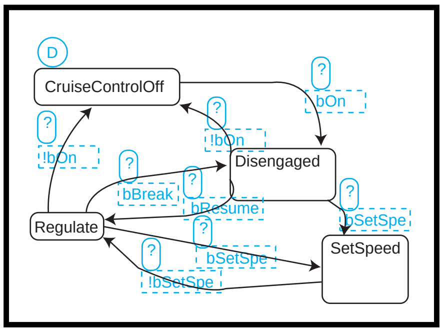


Figure 1—*In a conventional cruise-control state diagram, there is no state nesting. The cruise-control state machine maps input to output sequences.*

STATE-MACHINE CONTROL

State machines describe the reactive nature of many applications, including automotive. In particular, they describe the progression of the system with time, as inputs keep arriving into the system and outputs need to be produced by it.

For example, the state diagram in Figure 1 depicts a simple cruise-control machine, which changes states as inputs from other parts of the car are accepted.

Such state diagrams are well-known, well-accepted, highly visual, and intuitive. Their ability to describe finite and infinite sequences, combined with their visual appeal, has made them one of the most commonly accepted formalisms in the electronics industry.

State diagrams are easier to design, comprehend, modify, and document than a textual approach. But, they haven't changed much over the last 30 years, and they're limited when applied to modern reactive applications.

One problem is that state diagrams are flat. They can't handle modern top-down design and information hiding. Top-down design concepts require interactive software so the user can manipulate and browse complex designs.

Another disadvantage is that state diagrams are purely sequential, whereas applications are not. Modern state-machine controllers need to react to signals going to and coming from a plurality of entities in their environment. Also, a single state-machine controller might be responsible for many independent or partially independent conceptual tasks, thereby performing "conceptual concurrence."

Compensating for these limitations are statecharts, designed by David Harel [1]. While addressing the hierarchy and concurrence problems of state diagrams, statecharts retain the visual and intuitive appeal inherent to state diagrams.

The statechart in Figure 2 describes a more advanced cruise-control design than the state diagram of Figure 1. Note how the *CruiseControlOn* state encapsulates lower level states, thereby creating state hierarchy.

An important property of such hierarchical representation is that the high-level transition (labeled *!bOn*) from *CruiseControlOn* to *CruiseControlOff* fires regardless of the actual present state within *CruiseControlOn* and its descendants.

In fact, if the design is enhanced with more states within *CruiseControlOn* later in the design process, the high-level transition automatically encapsulates these newly added states, without any other explicit change to the design.

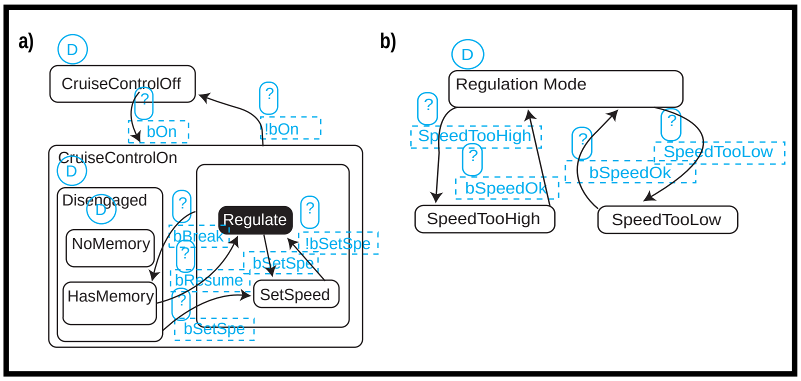


Figure 2a—*In this cruise-control statechart, state nesting exists within the* CruiseControlOn, Active, Disengaged, *and* Regulate *states. The* Regulate *state has nested states that are currently hidden from view.* ***b**—The contents of the* Regulate *state provide yet another example of state nesting.*

Note how the contents of the *Regulate* state are pushed onto a separate design page to provide more design space as we dive into lower levels of hierarchy. I'll discuss concurrence—the ability to describe multiple, simultaneous, or independent substatecharts—later when I extend the cruise-control example.

A HETEROGENEOUS EXAMPLE

Automotive applications often require that complex statechart designs be integrated with other types of computations, thereby creating heterogeneous (or hybrid) systems.

In my example, the cruise-control statechart of Figure 2 is extended to account for an event in the transmission box where the gear shifts from D (drive) to N (neutral), while cruise control is regulating (i.e., is in the *Regulate* state) the speed.

This event forces the statechart to change state to *Disengaged*. Indeed, it's a common problem in cruise-control–equipped vehicles that when the gear changes to N, the cruise-control logic attempts to accelerate, even though the transmission is not engaged. This action sharply increases the engine RPM.

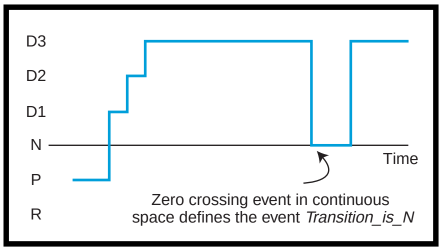


Figure 4—*This diagram illustrates a continuous waveform for the transmission box, from which the* Transmission_is_N *condition is generated.*

Figure 3 includes a simple modification to Figure 2, where the *Transmission_is_N* event causes a transition from the *Active* to *HasMemory* state. This transition disengages the cruise-control operation when the gear is set to N.

As illustrated in Figure 4, the *Transmission_is_N* event is generated from a continuous model, where the event occurs on a zero-crossing of the continuous signal. The interesting—and potentially difficult—aspect of this enhancement occurs with the real-world implementation of a statechart as a block of code and the progression of time.

Whether the statechart is implemented in C, C++, Java, or some other form of code, it will be invoked (i.e., called or activated) repeatedly in the real world. Each time it is invoked, it computes its new state or states (if the statechart has hierarchy or concurrence) given the previous state(s).

This repeated calling scheme can be periodic, as illustrated by the shorter arrows in Figure 5, or based on an event (e.g., every time a certain interrupt occurs).

There will always be periods of time between successive invocations of the statechart block of code in the real world. More often than not, the change in gear event (i.e., the gear shifting from D to N) occurs in some time slot between two successive invocations of the statechart code, as pictured in Figure 5.

There will be some period of time between the gear event and the next closest invocation of the statechart code that can use this information and disengage the cruise control. Let's denote this time period as an "unstable period."

Clearly, it's important that an event generated from the continuous representation of the transmission box trigger an invocation of the statechart code. Otherwise, the statechart of Figure 4 behaves improperly during its unstable period. This ability is the essence of a heterogeneous design, for simulation and implementation purposes alike.

The statechart in Figure 6 introduces concurrence. In this modified cruise-control example, there are two separate threads (depicted as dashed boxes)—the original *CruiseCntl* statechart and, simultaneously, the *Transmission* thread, which is entirely event based. *Transmission* changes

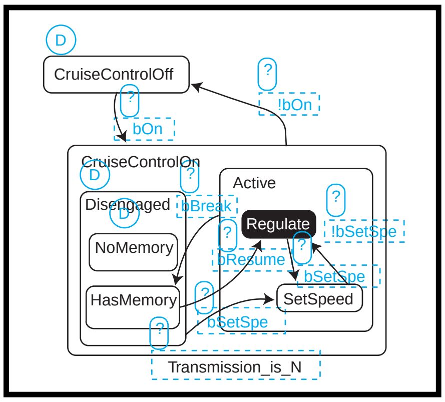


Figure 3—*This is the top-level enhanced cruise-control statechart. Note the* Transmission_is_N *transition from the* Active *state to the* HasMemory *state. The condition* Transmission_is_N *is generated by a continuous model.*

states as events from the continuous transmission model of Figure 4 occur.

Transmission constrains the allowable state changes in the transmission box. This thread performs a task that is mostly independent of the *CruiseCntl* thread, so it is described as concurrent.

However, some dependencies exist, such as the *Transmission* thread forcing *CruiseCntl* into the *Disengaged/HasMemory* state when the transmission's state becomes N. Visual synchronization—the ability to visually show dependencies between threads—is another powerful feature of statecharts.

HETEROGENEOUS DESIGN

As I mentioned, designing critical automotive control systems requires a heterogeneous environment. It's necessary to support high-fidelity process behavioral modeling, graphical software programming, design verification via simulation, and software specification and implementation via automatic code generation.

One helpful tool is BetterState, which aids in statechart design, code generation, and visual debugging. It supports all features of Harel statecharts, plus some extra capabilities like visual synchronization, visual priorities, critical regions, and mixing flowcharts and statecharts.

BetterState's automatic code generator produces code in C, C++, Java, Perl, Visual Basic, VHDL, Verilog HDL, Delphi, and special real-time OS (RTOS) code for embedded controllers.

As well, developers can use certain features to customize their code generator to write special-style code or code in another language. For example, a code generator that makes assembly code for the Motorola 68k processor is available.

Visual-debugging tools enable you to observe the statecharts' behavior in run time using state animation. This animation works even when the generated code is on a target embedded CPU.

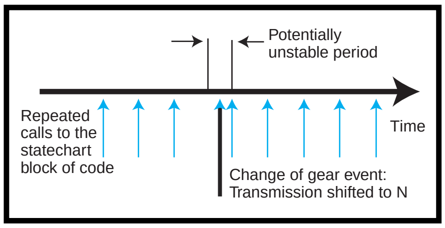

Potentially unstable period
Repeated calls to the statechart block of code
Time
Change of gear event: Transmission shifted to N

Figure 5—*A change-of-gear event happens in a potentially unstable period.*

To achieve the support goals mentioned above, I interfaced BetterState with MATRIX$_x$—a visual modeling, design, simulation, and implementation tool for large-scale control systems.

Each algorithm is described in MATRIX$_x$ and then automatically coded in C. The set of generated procedures is introduced in BetterState and activated as defined by the statechart. The combination of the MATRIX$_x$ algorithm and the statechart design creates the complete controller logic.

With the integrated environment, the design can be shared among team members without requiring any transfer and modification, which leaves time for verification and validation. This phase is first performed using the MATRIX$_x$ SystemBuild simulator.

The controller model can be tested against a process behavioral model also designed in MATRIX$_x$. Interactive animation tools can debug and interact with the design. Once the simulations execute satisfactorily, the automatic code generation can be performed.

The automatic implementation makes optimized code both from SystemBuild and BetterState. Concurrence and hierarchy describe a conceptually huge Cartesian product state space representing all combinations of states within concurrent threads anywhere in the state hierarchy.

However, the compact code generated by the code generator doesn't blow up state space at all. Rather, its size grows linearly with the number of transitions in the statechart. Time saved in code generation lets developers focus on software architecture and unit testing.

BENEFITS

Using such an heterogeneous environment for automotive control system design and implementation creates a work environment that facilitates workgroup communication, code reusability, and code-error reduction, while maintaining compact and efficient code.

More attention can be given to critical aspects of embedded software design such as software architecture and testing, which results in better quality and helps meet time-to-market agendas.

CODE GENERATION FOR RTOS

Many, if not most, real-time applications contain a reactive component that deals with sequences of inputs from and outputs to the environment surrounding the application. For example, a cellphone application requires a reactive component that deals with sequences of push-button events made by the user or with sequences of network commands received over the channel.

Such reactive components are almost always designed using a state-machine approach. Statechart and state-diagram code generation for a real-time OS needs to be robust in the presence of interrupts and other forms of reentrancy.

#110

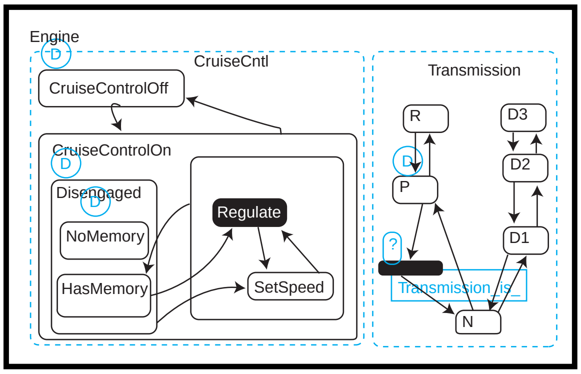


Figure 6—*This time, the Cruise Control statechart is augmented with concurrence (condition names are omitted for clarity). Note how* Cruise-Cntl *and* Transmission *are independent substatecharts in their own right.*

Typical code generators, whether for statecharts or conventional state diagrams, assume that the generated code (typically wrapped inside a function) completes executing before being called again.

This assumption is valid for simulation purposes. However, it doesn't hold when the generated code is invoked by interrupts or some other preemptive mechanism, where the current execution might be interrupted by another call due to a subsequent interrupt.

To address such concerns, BetterState offers a robust code generator tailored for pSOS and other RTOSs.

AN EMERGING STANDARD

Statecharts have been chosen as the de facto and de jure standards in many industries (e.g., SEMI, CASE/OOD, aerospace, and EDA). And recently, statecharts have been emerging as an important language in the automotive market as well.

In this article, I've shown how statecharts can be used in a heterogeneous design environment that combines statecharts and state-machine control with continuous-time control simulations and code generation.

It lets designers describe, simulate, and implement automotive applications such as engine control and antilock braking as well as complex cruise-control and body-logic applications. ■

Doron Drusinsky-Yoresh received his Ph.D. from the Weizmann Institute, Rehovot, Israel. He developed statechart CAD tools and DSP applications for Sony, and in 1993, he founded R-Active Concepts and developed BetterState. You may reach him at doron@isi.com.

REFERENCE

[1] D. Harel, "Statecharts: A Visual Approach to Complex Systems," *Science of Computer Programming*, **8**, 231–274, 1987.

SOURCE

BetterState, MATRIX$_x$
Integrated Systems, Inc.
201 Moffett Park Dr.
Sunnyvale, CA 94089
(408) 542-1500
Fax: (408) 542-1955
www.isi.com

I R S

407 Very Useful
408 Moderately Useful
409 Not Useful

FEATURE ARTICLE

Avi Cohen

Building Advanced Device Drivers for the MPC860

A new breed of 32-bit micros has arrived. Their complexity exponentially increases the demand on programmers. Using the MPC860 and DriveWay, Avi shows how to simplify the task of creating a tailored board.

More and more, we're seeing a new breed of 32-bit microprocessors for embedded applications. By adding, altering, and refining complex on-chip peripherals, vendors can provide cost-effective, high-performance derivatives for market niches.

Pushing functions like specialized telecommunications data management or LCD systems onto peripherals lets the embedded microprocessor manage them without tying up the core microprocessor or requiring extra silicon.

Combining multiple capabilities on a single chip greatly simplifies the final system. But, without proper microprocessor programming, you can't take full advantage of their functionality.

On-chip peripherals can consume up to 50% of the total silicon area and require hundreds of registers containing thousands of bits to be initialized. As complexity grows, interfacing application code to hardware becomes more time consuming and error prone.

As an example, consider the Motorola MPC860. It combines a PowerPC core capable of running at 40 MHz (upcoming versions will operate at 50 MHz) with an on-chip peripheral set handling everything from a design's DRAM interface to sophisticated serial devices that support protocols like Ethernet, HDLC, and T1/E1 time-division-multiplexed channels.

Numerous variants of the MPC860 offer a wide range of peripheral configurations. But, they all share a common set of logic external to the core PowerPC-based CPU.

This logic set includes an MMU, instruction and data caches, an interrupt controller for managing and prioritizing asynchronous events, and eight memory controllers that can interface to everything from the simplest ROM device to DRAM devices not even available yet.

Besides the core peripherals, variants contain combinations of SCCs (serial communications controllers) supporting many protocols (e.g. Ethernet, HDLC, and synchronous and asynchronous UARTs). There are also ports for I^2C, SPI, and PCMCIA, and most variants also contain two SMC (serial management channel) devices supporting less sophisticated serial protocols.

PERIPHERAL PROGRAMMING

The MPC860 is a complex, highly integrated embedded micro that places a significant burden on the developer to configure and control the peripherals without taking months to complete the task. Figure 1 shows how the peripherals interact with the core processor.

There's a basic set of peripherals that must be configured—the system setup, memory controller, timers, and boot code. As well, some peripherals are normally set up (e.g., SCC and SMC), and a host of them are sometimes set up (e.g., I^2C and SPI).

At a minimum, an application needs code to handle reset conditions and some external memory (ROM for program storage and RAM for data storage). As well, all applications need initialization of the on-chip interrupt controller.

Some global control parameters need to be defined (e.g., the location of the on-chip dual-port RAM shared between the PowerPC core and CPM), and so do the operation of some global clock signals.

To date, the most common solution for developing the appropriate software to interface the application code to the microprocessor's capabilities has been for in-house engineers to handcraft code. Some development teams take the "better to borrow than build" approach and integrate low-level control code provided by third parties or RTOS vendors.

A recently emerging option is to use development tools that automatically generate the device-driver code of this interface. In this article, I look at how one such tool, DriveWay-MPC860, simplifies the task of building and integrating a tailored board support package for a Motorola MPC860-based system.

DRIVEWAY

DriveWay-MPC860 is a Windows-hosted tool that, through an intuitive point-and-click interface, enables the user to configure the entire MPC860 to specific design requirements and then generate C and assembly source code that can link with the application.

This generated source code contains all the functions necessary to take the processor out of reset, initialize the entire device (including any OS parameters used by the application), and provide the application with a robust and well-documented API for accessing peripheral functions.

To shield the developer from the implementation details, the DriveWay configuration process works at the functional level, as opposed to the register level.

For example, the user interface doesn't ask whether to set the CRC bits in the PSMR register when configuring a channel for HDLC operation. Instead, the user can set a 16- or a 32-bit CRC calculation for HDLC messages, which DriveWay-MPC860 translates to the correct register mapping.

By generating the hardware and application code interface with DriveWay, the designer can concentrate on functionality and not implementation. While using DriveWay doesn't preempt the need to thoroughly understand the '860, it eases device programming.

It can also warn the user about contradictions among MPC860 resources. For example, most MPC860 I/O ports can be configured as general-purpose I/O or dedicated peripheral pins. Port C I/O pin 15 can be used for DMA request (DREQ), serial-port request to send (RTS), or general-purpose I/O.

One group of device-driver software engineers might write the DMA devices driver and use that pin, while another group uses the same pin for a serial-port device driver. DriveWay prevents such mistakes.

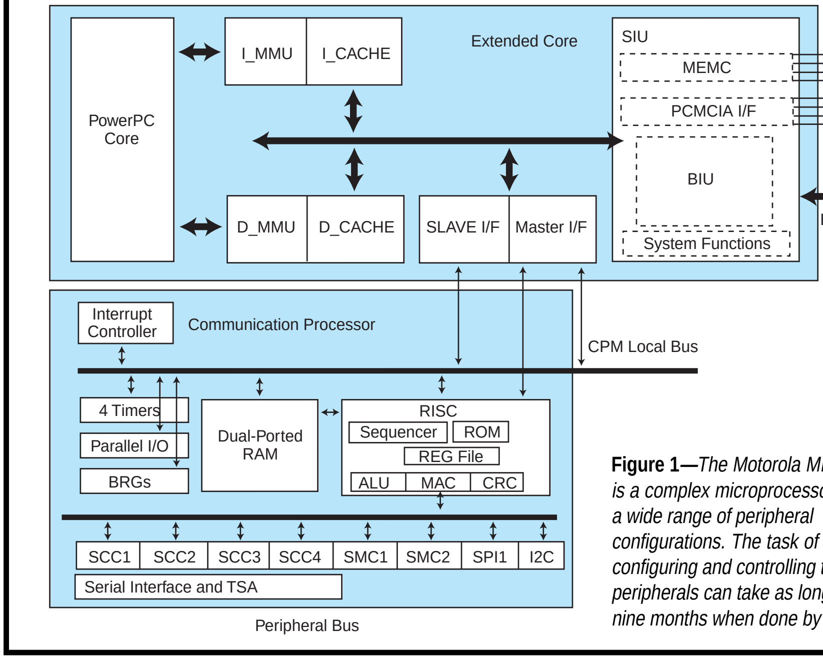


Figure 1—*The Motorola MPC860 is a complex microprocessor with a wide range of peripheral configurations. The task of configuring and controlling these peripherals can take as long as nine months when done by hand.*

The whole rationale of automating this step is that configuring and controlling the peripherals doesn't add value to the end product. It just enables the system to work. So, by reducing time spent on this phase, designers can focus on features that increase system value.

Two complex peripherals that must be configured are the memory controller and serial communication channel. These flexible peripherals support a wide range of options. Configuring them is key for optimal use of the embedded microprocessor.

One powerful feature of the MPC860 is its User Programmable Machine (UPM) support for external DRAM. Through a common programming approach, the microprocessor can support any DRAM that might be part of an embedded design.

An array of values can be loaded into on-chip dual-port memory, completely controlling the timing of the external RAS/CAS signals necessary to provide the refresh logic for external DRAM.

DriveWay makes programming this aspect of the MPC860 straightforward (see Photo 1), especially when used with Motorola's UPM860 tool for determining the values necessary to support a specific refresh behavior of an external DRAM device.

The designer can use UPM860 to map a refresh timing diagram onto an array of values to be loaded into the UPM and then use DriveWay-MPC860 to configure the boot code. Once code is generated, the software needed to glue the external DRAM to the MPC860 is complete.

DriveWay includes a library of UPM array of values for popular DRAM chips. When you select the chip's name, the generated code includes the UPM array of values for that chip. Listing 1 shows some of the boot assembly code needed to configure the UPM for a DRAM chip on Motorola's ADS860 board.

Due to the complexity of the protocols supported by the SCC channels, programming these devices is often the most difficult aspect of configuring an MPC860. The MPC860's CPM manages a set of buffer descriptors that must be configured properly by the initialization code.

There is also a set of registers for each SCC that is common to all of the protocols supported, as well as protocol-specific sets. All of these must be properly configured for optimal use of the microprocessor.

The issues that generally must be addressed when building a driver for an SCC channel include determining the operation mode to be established by the initialization sequence, defining an interface that enables application code to transmit and receive messages, and establishing the behavior of the system when asynchronous external events occur (e.g., a packet arrives along the communications port).

A number of parameters must be defined to ensure the correct operation of an SCC. A few months ago, Simon Napper discussed setting up an AMD 186 UART ("Writing Device Drivers for Embedded PCs," *INK* 86). Here, I examine how DriveWay documents and implements the selection.

To fully configure a UART using an MPC860 UART, the user must:

- define clock sources used by the SCC (the sources' clock rate is set in another function)
- define general-purpose I/O pins as dedicated SCC1 peripheral pins—TX (transmit), RX (receive), RTS (ready to send), CTS (clear to send), and DTR (data terminal ready)
- set SCC general-purpose mode register to operate in UART protocol
- set UART protocol-specific parameters (e.g., parity, data length, stop bits, etc.)
- set MPC860 internal RAM parameters of SCC1
- set buffer descriptors mechanism for reception and transmission
- reset UART protocol error counters
- initialize control characters and multidrop address tables (only when working in multidrop mode)
- set up general-events monitoring system and clear all previous event indications
- insert SCC1 interrupt service routine to the CPM handlers table

The code fragment in Listing 2 implements the second step—setting up the pins.

As you see, the documentation tells the user which signals will be applied to which pins. This information comes from the dialog selections made when configuring the UART. The code then sets the registers to meet the requirements. If the user wants to use the CTS signal, the code is modified to add it.

In this case, the mode of operation is NMSI (nonmultiplexed serial input). It's possible to multiplex several SCCs using the time-slot assigner supported by the '860. Ethernet and HDLC have different pins and transfer protocols, but they follow the same approach.

DriveWay abstracts the user away from coding details and focuses the design effort on what is required—not how to achieve it. Nevertheless, the code is documented thoroughly enough for you to walk through it and find out what was implemented and why.

RTOS AND COMPILER TOOLS

After the hardware interface is configured, the drivers must be configured to the run-time and development environments.

RTOSs are now common, and increasingly, commercial packages are being used rather than proprietary code. While there are many RTOSs to choose from, most share some common characteristics, including (apparent) concurrent task execution, a mechanism for prioritization, and primitives for passing messages and synchronizing tasks.

Most RTOSs are designed with a certain amount of hardware independence in mind. That is, the hardware on which the application and RTOS will execute has been abstracted.

For example, most require a periodic heartbeat timer that notifies the OS when a specified time interval (typically ~10 ms) has expired. Thus, certain OS services can be sensitive to elapsed time. Waking up a task at a specific time or letting certain blocking system calls time out are two reasons why an RTOS must know about the passage of time.

Once the driver's code is developed, it has to be integrated with the operating system. The combination of drivers

Listing 1—*This section of the boot code configures the memory controller for the DRAM on the Motorola ADS evaluation board.*

```
; Set up User Programmable Machine
   lis    r5,UPM_Initialize_Values@h
   ori    r5,r5,UPM_Initialize_Values@l
; User-Programmable Machine A—RAM Array
   li     r6,0x0000
   li     r7,0x0040
Write_UPMA_Loop:
   lwz    r8,0(r5)
   stw    r8,MDR(r3)
   stw    r6,MCR(r3)
   addi   r5,r5,4
   addi   r6,r6,1
   cmp    r6,r7
   blt    Write_UPMA_Loop
…
; UPM register array of values used for ADS860 board DRAM
; Single Read (Offset 0x0)
   .long 0xFFFFFF24, 0x0FF3CC24, 0x0FF3CC04, 0x0CF3CC04
   .long 0x00F3CC04, 0x00F3CC00, 0x37F7CC47, 0xffffffff
; Burst Read (Offset 0x8)
   .long 0xFFFEFF24, 0x0FF3CC24, 0x0FF3CC04, 0x08F3CC04
   .long 0x00F3CC00, 0x00F3CC0C, 0x0CF3CC44, 0x00F3EC08
   .long 0x03F3EC04, 0x00F3EC40, 0x00F3CC0C, 0x0CF3CC44
   .long 0x00F3EC00, 0x00F3EC04, 0x3FF7EC47, 0xffffffff
; Single Write (Offset 0x18)
   .long 0xFFFFFF24, 0x0fafcc24, 0x0FAFCC04, 0x08AFCC04
   .long 0x00AFCC00, 0x37FFCC47, 0xffffffff, 0xffffffff
; Burst Write (Offset 0x20)
   .long 0xFFFFFF24, 0x0FAFCC24, 0x0FAFCC04, 0x08AFCC00
   .long 0x07AFCC4C, 0x08AFCC00, 0x07AFCC4C, 0x08AFCC00
   .long 0x07AFCC4C, 0x08AFCC00, 0x372FCC47, 0xffffffff
   .long 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
; Refresh (Offset 0x30)
   .long 0xE0FFCC84, 0x00FFCC04, 0x00FFCC04, 0x0FFFCC04
   .long 0x7FFFCC04, 0xFFFFCC86, 0xFFFFCC05, 0xffffffff
   .long 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
; Exception (Offset 0x3c)
   .long 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
```

and initialization code integrated with the RTOS is the board support package (BSP).

Along with the standard services for each peripheral, the BSP developer creates code that initializes each device and installs an interrupt service routine that notifies the RTOS when an event occurs. When integrating driver code with an RTOS, the BSP developer must keep a number of issues in mind:

- how does the boot code initialize the operating system?
- what RTOS services must be available when initializing the devices?
- where is control of the interrupt vectors based—the driver or the RTOS?
- how are the device drivers glued to any I/O services provided by the RTOS?

Using an automation tool to configure and build a BSP shields the developer from many of these issues.

Boot code that not only initializes the microprocessor but also invokes the proper sequence for bringing up the RTOS can be automatically generated. It's possible for the RTOS to maintain control of the entire interrupt system if the proper API has the RTOS install the interrupt service routines required by the device drivers.

The MPC860 has a complex interrupt system structure. For example, SCC2 requests an interrupt when a data buffer is received. This request is sent to the MPC860 CPM interrupt controller.

If the priority of SCC1 is high enough and SCC1 interrupts are not masked, then the interrupt request is sent to the MPC860 SIU (system interface unit). If the CPM interrupt priority is high enough and the CPM interrupt request is not masked, then the interrupt request goes to the MPC860 core as an external interrupt request. If the interrupts are not masked in MPC860's core, the interrupt request will be served.

Listing 2—*This code fragment, generated by DriveWay, configures pins 14 and 15 for RXD (Receive Data) and TXD (Transmit Data), respectively, and sets up BRG1 to be TCLK (Transmit Clock) and RCLK (Receive Clock).*

```
/* SCC activation sequence:
*    Initialize SCC1 in Uart mode: Scc1UartInit();
*    Enable SCC1: SccEnable();
*    Transmit data: SccTxBuffer() or SccTxFrame()
*    Disable SCC1: SccDisable(); */
void Scc1UartInit(void)
{
  SCC_P SccPtr;        /* Pointer to SCC Configuration Registers */
  SCC_UART_P SccUartPtr; /* Pointer to SCC UART Protocol Config
                              Registers */
  SCC_PARAM_P SccParamPtr; /* Pointer to SCC Parameters RAM
                               Registers */
  BD_P BdPtr;          /* Pointer to Buffer Descriptor */
  U16 * AddrTbl;       /* Pointer to Address Table */
  U16 * CtrlCharTbl;   /* Pointer to Control Characters Table */
  register int i;      /* General index */

 /* SCC 1 is connected directly to the NMSI pins
  - RXD signal supported by Port A pin 15
  - TXD signal supported by Port A pin 14
  - RTS signal not supported
  - CTS signal not supported
  - CD signal not supported
  - DTR signal supported by Port D PIN15
  - Both TCLK and RCLK signals supported by BRG1 */
 Quicc->Sicr |= ((SCC_NO_GRANT_SUPPORT   |
                  SCC_NMSI_INTERFACE     |
                 (BRG1_CLK_SRC<<3)       |
                  BRG1_CLK_SRC));
 /* Set TXD/RXD signals (Port A pins 14-15) */
 Quicc->PortA.Ppar |= ( HALF_WORD_BIT14 | HALF_WORD_BIT15 );
 /* Set DTR signal (Port D pin 15) */
 Quicc->PortD.Pdir |= (U16)PIN15;
 Quicc->PortD.Pdat |= (U16)PIN15;
}
```

Most RTOSs include support for this complex structure via setting up the interrupt table and resolving the interrupt source, as well as the peripherals' interrupt handlers and general interrupt services (e.g., insert an interrupt handler in the interrupt table, enable and disable interrupts, etc.).

The MPC860 includes an event and error report system that can be used for interrupt handling. Different applications need to be notified about different events or errors, and this task is accomplished during the interrupt-handling process and using the MPC860 buffer descriptor (BD) mechanism.

For each peripheral, mode, or protocol, a set of bits located in the BD reports an error or event.

Most RTOSs supply BSPs that include a few functions to enable the user to get the RTOS running and to aid the debugging process. For example, the MPC860 SMC1 is used by most of the RTOS as a debug channel.

However, this package is incomplete. The reset of the MPC860 peripherals' interrupt handlers, for example, is not supplied. Some RTOSs don't include any part of the interrupt system, so it becomes the designer's responsibility.

Some applications don't need any RTOS environment (e.g., a testing system or diagnostic program). For these cases, the user must provide a stand-alone system that includes all the necessary boot code, interrupt system, memory management, make files, and peripheral device drivers.

In addition to the OS environment, a code-generation tool must be aware of the compiler environment in which the code will be built. For optimal driver performance, it's often necessary to take advantage of some of the extensions to ANSI C provided in most toolkits for embedded software development.

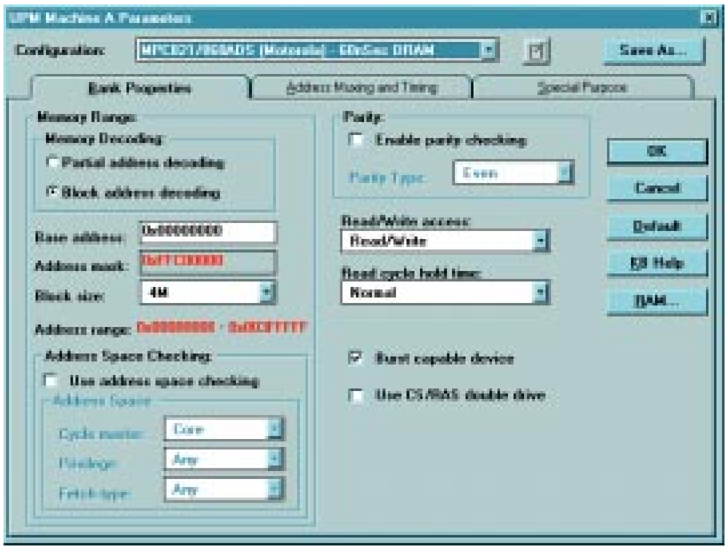
Extensions such as inline assembly code and use of a "packed" qualifier for structure definitions are common-place in most embedded tools. But, all use a slightly different syntax.

For example, the Diab Data `#pragma pure_function` promises that the function does not modify or use any global or static data. This information helps the compiler make a few assumptions on the nature of the code and generate better and faster code.

The `inline` keyword provides a way to replace a function call with an inline copy of the function body, reducing the time needed to save registers and set up the stack before and after the function call.

All these details must be correctly set. The possible combination of compiler tools, RTOS, and the functional requirement of the system means that no prepared driver set will ever be optimal.

DriveWay 3DE enables users to specify what they want using dialog selections. Users must understand which options are required, but they don't need to understand the coding details.



Configuration: MPC821/860ADS (Motorola) - 60nSec DRAM

Memory Range:
Memory Decoding:
Partial address decoding
Block address decoding [✓]
Base address: 0x00000000
Address mask: 0xFFC00000
Block size: 4M
Address range: 0x00000000 - 0x003FFFFF

Address Space Checking:
☐ Use address space checking
Address Space
Cycle master: Core
Privilege: Any
Fetch type: Any

Parity:
☐ Enable parity checking
Parity Type: Even

Read/Write access: Read/Write
Read cycle hold time: Normal

☑ Burst capable device
☐ Use CS/RAS double drive

Photo 1—*DriveWay presents the user with dialog boxes to configure the memory controller on the MPC860. DriveWay uses the selections made in the dialog boxes to generate the code desired.*

DEBUGGING CHALLENGES

The design and coding of device drivers for these advanced processors is always a challenge. But, debugging them is usually the most difficult task.

There are many source-level debuggers (SLDs) used in embedded software development that make use of a "monitor" on the target board to interface with a host-based tool. But, a monitor is worthless unless there is already boot code to perform minimal initialization of the board and a device driver in place for communication between the monitor and SLD.

To solve this problem, the MPC860 includes a new system called Development System Interface. This system uses a dedicated development serial port, which is a relatively inexpensive port that doesn't need the usual system interfaces, memory, boot code, and so on.

Hardware assist solutions (e.g., emulators and logic analyzers) can provide the debugging tools necessary to troubleshoot these sorts of problems. But such solutions tend to be expensive, and they're not always available. Often, the only hardware aid a developer has are onboard LEDs that can be toggled to indicate where a problem might be.

With a complex microprocessor like the MPC860, extra debugging problems can arise due to subtle interactions among the peripherals. When so many devices share the limited dual-port memory that channels communications between the PowerPC core and the CPM, it's easy to create code that results in the driver for one device corrupting the operation of another device.

Of course, debugging can absorb enormous amounts of time. Although using DriveWay doesn't preclude the need for debugging, the fact that the drivers are already thoroughly tested and the application-code interface is well-documented eliminates two sources for bugs and reduces the time needed to get through this difficult phase.

OVERCOMING OBSTACLES

New, complex 32-bit microprocessors for embedded applications have created both opportunities and obstacles for today's embedded-system designers.

The opportunity comes from the breadth of solutions offered in a single-chip package. Power management, memory control logic, a powerful CPU, and sophisticated I/O controllers on a single device offer great advantages.

But, managing this complexity is probably the most difficult issue facing the embedded-system developers. Design tools help address these issues.

There is growing interest and activity in the embedded market, demonstrated by increased attendance at embedded conferences and the growing number of embedded microprocessors and tools available.

New tools and technologies are arriving on the market to increase the leverage of the design engineer developing embedded systems.

As complexity continues to escalate rapidly, designers must select their tools carefully because the tools will increasingly dictate what can be achieved on any given project. ■

Avi Cohen is manager of the Embedded Systems group at Aisys. Prior to joining Aisys, he was a software engineer with R.N.D. Switches and Routers. You may reach him at avi_cohen@aisys.co.il.

SOURCES

DriveWay
Aisys, Inc.
4633 Old Ironsides Dr., Ste. 105
Santa Clara, CA 95054
(408) 327-8820
Fax: (408) 327-8830
mkt@aisysinc.com
www.aisysinc.com

MPC860
Motorola
MD: OE216
6501 William Cannon Dr. W
Austin, TX 78735-8598
(512) 895-3823
Fax: (512) 895-8807
michael_shoemake@risc.sps.mot.com
www.mot.com/SPS/RISC/netcomm/prod/eppc/MPC860.html

I R S

410 Very Useful
411 Moderately Useful
412 Not Useful

EMBEDDED PC

JANUARY 1998

Photo courtesy of Ampro Computers, Inc.

PC/104 CPU MODULE

The **AT/3I Communication Engine** is a low-power, PC/104-compliant CPU module based on a '386SX processor running at 33 MHz. Its integrated Ethernet, PCMCIA, and serial ports make it an ideal choice for a low-cost system with connectivity features.

The AT/3I features two RS-232 serial ports (one could be configured as RS-422/-485), a bidirectional parallel port, and AUI NE2000 Ethernet. It also has a PCMCIA slot, AT keyboard slot, watchdog timer, real-time clock, embedded FlashBIOS, and 15 general-purpose DIO lines. The module supports different operating systems like DOS, Windows, QNX, pSOS, and other RTOSs.

The PCMCIA interface is intended as a general-purpose communication port for high-speed V80 modem, ISDN, Token Ring, IBM 3270/5250 terminal emulation, and video-capture cards. In addition to its 2 MB of DRAM, the module integrates a solid-state disk that can be used to store the operating system, user program, and data files with a maximum capacity of 24 MB.

The AT/31 sells for **$595** in 1–10 quantities.

EuroTech srl
Via Linussio 1 • 33020 Amaro (UD) Italy
+39 433 486258 • Fax: +39 (0) 433 486263
pc104@eurotech.it • www.eurotech.it **#510**

EMBEDDED DEVELOPMENT TOOL

POWERplant EDE provides an integrated development environment by combining leading embedded development tools with Microsoft's Developer Studio. The capabilities of Developer Studio are extended to compile, link, and debug embedded applications using the best cross-development tools. These capabilities, combined with Nucleus MNT (a Windows NT-based prototyping environment), provide a full-featured embedded development environment.

The concept behind POWERplant EDE is simple. First, build and prototype a system using Nucleus MNT. This allows the use of Microsoft Visual C++ and supporting tools to develop and test the code before cross development. Once the prototype application is refined, by changing projects, the user can build, download, and debug an application within the Developer Studio environment using cross-development tools.

In addition to Developer Studio's edit, build, and test features, project-management, class-manager, and browser features are available. Code management and development tools can also be added.

POWERplant EDE is preconfigured with an initialization file that is read into a configuration program. This file contains the common directories and paths for the toolset being used, as well as all the compiler, assembler, librarian, linker, and locator command lines and switches that were used when Nucleus was built. For a different configuration, the user enters the configuration tool and makes the necessary changes (e.g., add or remove switches, add libraries, relocate directories, etc.).

Licenses for POWERplant EDE are **$1295** per seat.

Accelerated Technology, Inc.
720 Oak Circle Dr. E.
Mobile, AL 36609
(334) 661-5770 • Fax: (334) 661-5788
sales@atinucleus.com • www.atinucleus.com

#511

NouveauPC
edited by Harv Weiner

100-MHz EMBEDDED CONTROLLER

AMD has introduced a 100-MHz version of its Élan microcontroller family. The 32-bit **ÉlanSC400** combines the Am486 CPU with an integrated memory controller, PC/AT system logic, and essential mobile computing peripherals. Virtually all the peripheral logic required for a PC/AT-compatible system is on a single chip, and DOS, Windows CE, and other major ’*x*86-compatible RTOSs are supported.

In addition to the CPU, the ÉlanSC400 microcontroller features an integrated power management unit (PMU), five phase-locked loops that generate all system clocks from a 32-kHz watch crystal, and fully static design for maximum battery life. In addition to power management, the chip contains an LCD graphics controller, dual PC Card controller (PCMCIA 2.1 and ExCA compliant), 16C550-compatible UART, EPP-compatible parallel port, and an IrDA infrared port.

The chip’s complete memory controller supports 64-MB EDO and FPM DRAM. A complete ROM controller supports ROM and flash memory. A 16-bit ISA-bus controller and CPU Local-bus access (VL type) are also provided.

The ÉlanSC400 is priced at **$55.65** in quantity.

AMD
One AMD Pl.
Sunnyvale, CA 94088-3453
(408) 732-2400
Fax: (408) 732-7216
www.amd.com

#512

PC/104 DATA-ACQUISITION CARD

The **AIM16-2/104** is a high-speed (200 kHz) 16-bit data-acquisition card that features 16 single-ended or 8 differential analog input channels, 16 lines of digital I/O, flexible triggering options, direct memory access (DMA), and interrupt operation with a standard 4K × 16 FIFO. Applications for the card include medical imaging, spectroscopy, guidance systems, and automated test equipment.

The AIM16-2/104 provides 85 dB of spurious-free dynamic range (SFDR) with user-selectable inputs of 10, 5, 2.5, and 1.25 V, and it offers factory-installed bipolar and unipolar options. The board incorporates proven high-frequency layout techniques, including short, guarded signal paths and separate power and ground planes to ensure noise immunity. An onboard DC-DC converter powered by a single +5-V supply provides noise isolation from the system switching power supply.

The PC/104-compliant card is priced at **$625**.

Analogic Corp.
8 Centennial Dr.
Peabody, MA 01960
(978) 977-3000
Fax: (617) 245-1274
www.analogic.com

#513

NouveauPC

'586 CPU CARD

The **VL-586-1** is a DOS-compatible computer that combines standard DOS/Windows hardware with customized BIOS firmware and an STD/STD32 Bus interface. Its PC/104-*Plus* expansion site enables a PC/104 or PC/104-*Plus* module to be stacked directly onboard. This setup enables the use of high-speed video modules and "local" I/O expansion in systems using multiple processor cards.

The board features a 133-MHz AMD 5x86 processor that does not require an on-chip fan or special cooling. It accommodates 4–32 MB of DRAM, 2.5 MB of flash memory, and 512 KB of battery-backed SRAM, enabling infinite read/write cycling (unlike flash memory) with full power-fail protection. The board includes floppy, IDE, and AT keyboard interfaces, PC-compatible LPT port, two COM ports, real-time clock with CMOS RAM, interrupt controller, DOS counter/timer channels, and DMA controller. Built-in multiprocessor arbitration and a watchdog timer/reset circuit are also featured.

The VL-586-1 sells for **$676** in quantity.

VersaLogic Corp.
3888 Stewart Rd. • Eugene, OR 97402
(541) 485-8575 • Fax: (541) 485-5712
www.versalogic.com **#514**

Scott Lehrbaum

Year 2000 and Embedded PCs

The year-2000 phenomenon incites many hyped predictions of people trapped in elevators, loss of infrastructure, never mind millions of dollars. According to Scott, though, all embedded PCs need is a software patch. Read how it's done.

The rapid proliferation of computers in the last thirty or more years has been largely driven by an increasing level of performance and sophistication in both business and home systems.

Yet, the astonishing pace of the computer revolution could not have been achieved without a strong vendor commitment to the idea of backward compatibility. This enables customers to leverage off their previous investments in hardware and software but still make immediate use of the latest in computer technology.

While this strategy has accelerated both the takeover of computers and the development of advanced computer technologies, it has occasionally produced some unfortunate side effects. By far, the most infamous of these is the year-2000 problem, also known as the millennium bug or Y2K.

At first glance, the issue seems fairly innocuous. The year is represented in hardware and processed in software as a two-digit value.

But, the year-2000 problem poses a real threat to many of our most critical electronic systems. Everything from power plants, air-traffic control, and communications systems to health care, financial institutions, and government agencies could experience catastrophic system failures as a direct result of the year-2000 problem.

How could modern society be brought to its knees by two lousy digits?

First, the systems controlling most of these critical applications are old-style mainframe computers. They can't be upgraded with new hardware, and they require extensive and extremely costly software changes to eliminate flaws.

Secondly, in many cases, the application software relies heavily on time and date information to schedule critical system events. A bad date can cause these events to be initiated out of sequence or at the wrong times, potentially with disastrous consequences.

A lot of other application programs may simply quit working, shutting down or crashing the systems they control. If that happens to a mainframe system running a power-plant, phone-company, or air-traffic control system, the effects of the year-2000 problem could be devastating.

MILLENNIUM BUG UNMASKED

The year-2000 problem is rooted in the use of a two-digit value to store and process the year. Early mainframe designers had a good reason for building in such an obvious limitation—dropping two digits meant valuable savings in component size and manufacturing costs.

In the intervening years, new computer systems have been designed with the same limitation, for the equally good reason of maintaining compatibility with earlier software. Despite the fact that the original rationale has long been obsolete, the tenets of backward compatibility have consistently won out over alternative, more advanced clock designs.

Let's look at how the year is represented on systems that support only a two-digit year. For example, the year 1972 is stored as 72, and 1999 is 99.

But, what happens when the year rolls over to the next century? The obvious answer is that it reverts to 00, but how does software interpret that value?

In theory, if the software was aware of the two-digit year limitation, it could calculate dates correctly for 100 years by using a particular year as a baseline—logically, the year the software was written. Lower numbered years could then be interpreted as occurring after the century rollover, as illustrated in Figure 1.

This simple approach to managing the century issue in software is now being used in some year-2000–aware BIOSs and application programs. But from a practical standpoint, there's a lot more to this problem than how software interprets two-digit years.

THE PROBLEM IN PCs

Two missing digits might seem easy enough to cope with, but the question of year-2000 susceptibility in PCs is surprisingly complex.

In fact, several different design errors and limitations compose the year-2000 problem. They have accumulated over the nearly twenty-year history of PC hardware and software design and are all related to the original issue of a two-digit year.

In the following sections, I outline four key elements to the problem in the same order that they might come into play during system startup and operation (see Figure 2).

REAL-TIME CLOCK

The first desktop PCs introduced by IBM—the PC and PC/XT—didn't include an onboard real-time clock. It wasn't until the first AT that battery-backed timekeeping became standard.

The device chosen by IBM—the Motorola MC-146818—maintained the long-held tradition of storing the date as a two-digit value. Thus, with the decision to use the Motorola device, IBM designed in the millennium bug as a standard feature on its desktop systems.

Vendors of IBM-compatible clones have historically duplicated the IBM designs with few architectural changes. This decision makes sense for the same reason that using two-digit real-time clocks in new computer designs did. Maintaining 100% compatibility with existing platforms maximizes the value and benefit of new hardware purchases.

Unfortunately, it also means clone vendors hopped right on the millennium-bug bandwagon, incorporating the same or equivalent time-keeping devices in their own desktop designs.

BIOS

To account for the lack of a century indicator in the real-time clock, an IBM firmware engineer came up with the idea of reserving an unused byte in the real-time clock's memory (also used for CMOS setup data) to store the current century value.

This clever trick could have saved a lot of future heartache if it had worked as intended. However, the idea fell flat because there wasn't the capability to either detect a century rollover or increment the century counter at the start of a centesimal year. And as with the real-time clock design, clone manufacturers essentially duplicated the BIOS code, flaws and all.

WHAT DOS DOES

When DOS is first loaded into memory and executed, it requests the time and date from the BIOS to initialize its own internal clock. The DOS clock is automated via the system-timer tick function and is independent of the hardware real-time clock.

To maintain platform independence, DOS never accesses the real-time clock directly, relying on the BIOS to provide this service. Unfortunately, this technique ensures any errors not caught by the BIOS directly affect the DOS time and date.

The problem with DOS is in the way it handles—or more pointedly, fails to handle—a bad year value returned by defective BIOS code. If the year rolls to 00 on a system whose BIOS is not year-2000 enhanced, the year is returned as 1900.

But as far as DOS is concerned, January 4, 1980 (a date that probably has some significance in the history of DOS) is effectively the beginning of time. If it gets an earlier date, DOS discards it as invalid and resets to 01-04-1980.

DOS doesn't know about the century limitations of the BIOS or real-time clock. Therefore, it has no reason to try to distinguish a century rollover from an actual bad date, which might result from a real-time clock failure or dead back-up battery.

And, there are similar problems with other commercial OSs, including Windows 3.1, Windows 95, and Unix derivatives.

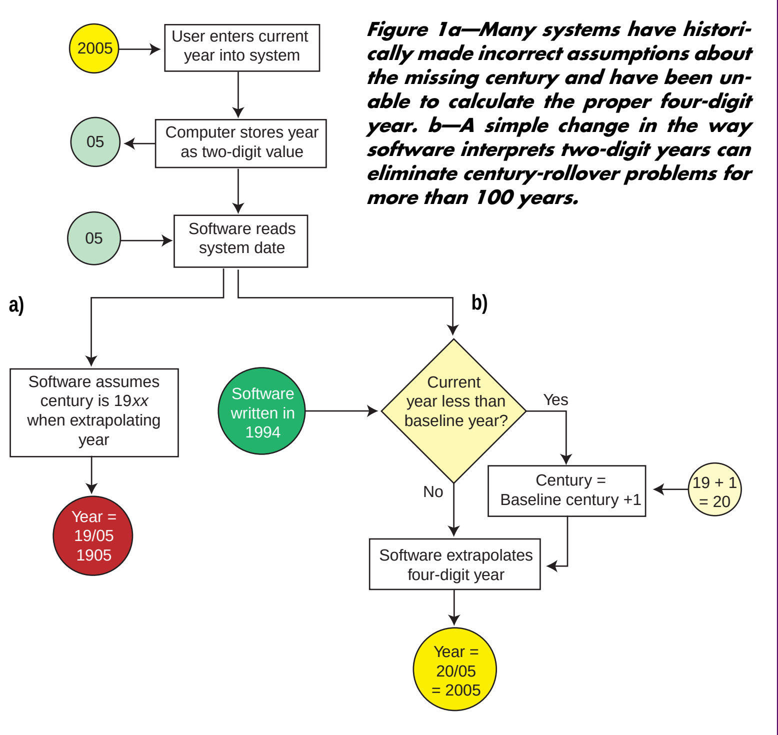

2005
User enters current year into system
05
Computer stores year as two-digit value
05
Software reads system date
a)
b)
Software assumes century is 19xx when extrapolating year
Year = 19/05 1905
Software written in 1994
Current year less than baseline year?
Yes
No
Century = Baseline century +1
19 + 1 = 20
Software extrapolates four-digit year
Year = 20/05 = 2005

Figure 1a—Many systems have historically made incorrect assumptions about the missing century and have been unable to calculate the proper four-digit year. b—A simple change in the way software interprets two-digit years can eliminate century-rollover problems for more than 100 years.

APPLICATION SOFTWARE

Commercial applications and user-designed software can be the most unpredictable factors in assessing system vulnerability to year-2000–related problems.

The BIOS and operating system generally take a passive role in managing system time- and date-keeping services. In contrast, application software depends heavily on accurate time and date information and uses it in many calculations and decision-making processes.

For example, application software in a banking system calculates and charges interest on a loan balance based on how much time has passed since the last charge was assessed. Computers controlling assembly lines use the time and date to schedule regular maintenance shutdowns.

In either system, how well the software handles the century rollover may determine the fate of the company. If the code is error free or has been cleaned of year-2000 flaws, the company should breeze through the changeover without interruption.

But if not, the company could be hit with massive losses in revenue and productivity and may have to shut down for long periods of time to make necessary repairs. For this reason, the vulnerability of application software is the single most urgent question organizations must answer to determine their year-2000 survivability.

MANAGING THE PROBLEM

Operating systems and application software normally use BIOS interrupt services to get time and date information from the real-time clock or to change its contents. Thus, the system BIOS is in an excellent position to eliminate year-2000 problems at the source.

The code in Listing 1 is similar to that used in the Ampro BIOS interrupt 1Ah `Get Real Time Clock Date` function. It shows a simple but effective way of making a PC/AT BIOS year-2000 immune.

The key to this patch's effectiveness lies in the fact that OSs generally request the system time from the BIOS during bootup in order to start their own internal clocks. This process gives the BIOS a chance to check and update the century counter each time the system reboots. It also enables updates to occur during regular operation on systems that have been left on since before the century rollover.

This capability could be critical for a lot of black-box embedded-PC applications which are never powered off or rebooted. At any time, the application software can make a call to the BIOS `Get Real Time Clock Date` function to assure that its current year and century information are properly updated.

This simple enhancement to the original IBM interrupt 1Ah code can effectively eliminate all BIOS-related year-2000 problems. Yet remarkably, BIOS vendors didn't begin to incorporate comparable fixes until the early to mid 1990s.

Older model PC systems present more of a challenge because there probably won't be a year-2000–immune BIOS available for the down-rev hardware. This problem is especially significant in the embedded market because usable product lifetimes are typically much longer than other types of PC systems. The population of embedded PCs more than a few years old that are still in use today is much higher than comparably aged desktop or notebook systems.

Despite the lack of BIOS upgrades, it should still be possible to shore up the defenses of older embedded-PC compatibles by using a device driver that mimics the functionality of the improved BIOSs.

One such driver is provided by Ampro for use on earlier generation AT-class products. It chains into the INT 1Ah service routine and monitors the century value returned by `Get Real Time Clock Date`.

If a rollover is detected, it increments the century and calls `Set Real Time Clock Date` to update the RTC. It also updates the DOS date in the event of a century rollover.

Such drivers provide the same protection against year-2000–related problems as an enhanced BIOS and can be installed in older systems with a simple software upgrade.

If you can patch your systems with a year-2000–immune BIOS or device driver, most of your problems will be solved. In particular, operating systems should behave well as long as they are provided with correct date information.

Even if no BIOS fix or driver is available, the latest versions of just about all operating systems now handle bad dates elegantly. If you suspect your OS version is susceptible to year-2000 problems, contact the vendor for more information and to arrange an upgrade to the latest release.

Regardless of how well your BIOS and operating system can weather the millennium change, there's no guarantee that the commercial or custom-written software that runs in your system is impervious to century defects. Unfortunately, a hardware or OS vendor can't do much to protect against software problems.

If you're using commercial off-the-shelf software in your application, you should

Listing 1—This software patch should protect PCs against century-rollover problems until the year 2100. It assumes 1980 to be the earliest possible year and automatically increments the century if a lower numbered year is detected.

```
cent_adr      equu   32h            ; RTC century address
year_adr      equ    09h            ; RTC year address
nmi_stat      equ    00h            ; always leave nmi enabled
                                    ; make 80h for nmi disable
;Return real-time clock year as four-digit BCD value in CX
;Update century if rollover is detected
Get_RTC_Year proc
              cli                   ; clear interrupts
              mov    al,cent_adr    ; load addr of RTC century
              or     al,nmi_stat    ; set desired nmi status
              out    70h,al         ; write century index to RTC
              in     al,80h         ; wait a microsecond or two
              in     al,71h         ; read century value
              mov    ch,al          ; save in CH
              mov    al,year_adr    ; load addr of RTC year
              or     al,nmi_stat    ; set nmi status
              out    70h,al         ; write year index to RTC
              in     al,80h         ; wait
              in     al,71h         ; read year value
              mov    cl,al          ; save in CL
              cmp    cx,1980h       ; 1980 earliest possible year
              jb     adjust_cent    ; if below, it rolled over
done:         sti                   ; restore h/w interrupts
              ret

adjust_cent:  mov    ch,20h         ; increment the BCD century
              mov    al,cent_adr    ; load addr of RTC century
              or     al,nmi_stat    ; set nmi status
              out    70h,al         ; write century index
              in     al,80h         ; wait
              mov    al,ch          ; new century value
              out    71h,al         ; update RTC
              jmp    short done
Get_RTC_Year endp
```

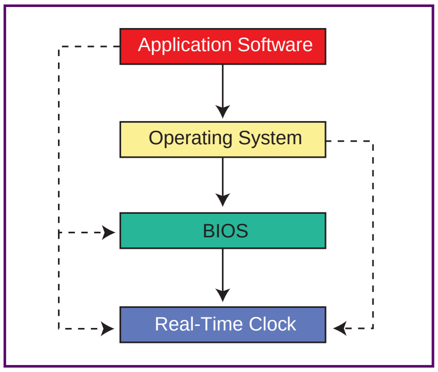


Figure 2—If time information were passed from one software layer to the next in a linear fashion (center arrows), century management would be fairly straightforward. But, complexity is added by the fact that this convention isn't always followed by OSs and application software (dotted arrows).

probably check with the software vendor regarding possible year-2000 glitches. Some of the most popular and widely used application programs have been found to contain bugs, although in most cases, the problems are relatively harmless.

A word of warning, though. Microsoft recommends against advancing the clock to 11:59 P.M. on 12-31-99 and then waiting to see what happens. Their advice suggests that the results of such tests could be unpredictable.

In most cases, software vendors should be able to identify year-2000–susceptible products and offer upgraded versions.

Custom software can be more of a challenge, depending on whether the engineer who wrote the original code is still around or how well the source is documented. Companies must evaluate proprietary code on a case-by-case basis to determine if corrective measures may be required.

If you need to modify the software in systems that are already installed in the field—to replace buggy code, upgrade the BIOS, or install a year-2000 software patch—there may be ways to minimize the effort needed to perform the update. This situation depends entirely on the system configuration and the capabilities of the specific product.

It may be worthwhile to discuss your options with your board vendor and strategize on the best approach for upgrading those systems. You may need to dispatch service personnel to the installation sites or have the systems returned to your facility for the required updates.

One final note on the year-2000 problem should be made relative to XT-class embedded PCs. Recall that IBM didn't include a real-time clock with desktop PCs until the introduction of the first AT-class system.

However, because battery-backed timekeeping is an important requirement in embedded systems, some vendors provide onboard or plug-in real-time clock options for their XT-class CPU products. These devices are necessarily different from the standard AT RTC because the XT architecture doesn't support the required interrupt channel (IRQ8).

Thus freed of the burden of backward compatibility, these devices typically provide full four-digit year counters, rendering them impervious to the sting of the millenium bug.

THINKING AHEAD

Considering the widespread and potentially devastating effects of the year-2000 problem, it's no surprise we're wondering how well embedded PCs will weather the century changeover.

The good news is that PCs in their many diverse form factors are much less likely to suffer from ill effects than mainframe computers and other predecessors. These newer systems are highly upgradable, permitting easy substitution of flawed hardware or software components with backward-compatible, year-2000–immune replacements.

Even so, there are a lot of factors to consider in evaluating the immunity or vulnerability of your embedded PC.

Evaluate each system thoroughly for possible weaknesses on all levels—hardware, BIOS, and software. And, take immediate corrective actions to ensure that all problems are resolved before the fast-approaching millennium deadline. EPC

Scott Lehrbaum has been at Ampro Computers for nearly nine years, where he has held numerous positions ranging from Software Engineer to Applications Engineering Manager. He has done extensive research into the year-2000 issue and designed software patches similar to those described in this article. You may reach Scott at slehrbaum@ampro.com.

IRS

413 Very Useful
414 Moderately Useful
415 Not Useful

Raz Dan &
Stefanie Hein

Flash-Disk Building Blocks

Tradeoffs in the Make vs. Buy Decision

Customer needs—real and perceived—drive the market. And, right now, you should be Web enabled. The result? You need greater mass storage. Raz and Stefanie guide you through the decision of buying a solution or making your own.

Embedded systems have moved from simple controllers with small applications to high-end processors that run 32-bit operating systems.

The driving force behind this change is customer requirements, both real and perceived. Today, it seems that even if you want to build a better mousetrap, it has to be Web enabled to be interesting.

One of the changes that's required to support these new features is the addition of large amounts of local storage. A 128-KB EPROM is no longer enough, yet products still have to meet many of the old requirements (e.g., low power consumption, light weight, small form factor, harsh environments, low cost, etc.).

In this article, we look at some of the mass-storage options available and their implementation tradeoffs.

The obvious solution for high-capacity storage and low cost is a magnetic hard disk. But, this solution is neither small nor lightweight, nor is it suitable for low-power devices. In addition, hard disks can't withstand the harsh environmental conditions that many embedded designs are subject to.

One solution that meets all the requirements is a solid-state disk drive. Many articles have been written about the different types of solid-state disks available, but the bottom line is that today, and in the upcoming years, the best technology is flash memory.

This has led to the development of a new class of disk drives—flash disks. Flash memory has many important characteristics—high density (with an aggressive road map), zero-power data retention, and reliable data storage.

Photo 1—DiskOnChip 2000 is a 32-pin DIP single-chip bootable flash disk with hard-disk compatibility, high performance, cost-efficiency, and extreme reliability. It is offered in capacities of 2–72 MB with future models having 144+ MB. All this in a package smaller than a matchbox.

Flash memory is also replacing ROMs in traditional applications (e.g., code storage), so there's a huge market for these components. The technology is attractive to many semiconductor manufacturers, which means they'll continue to invest in research and development and improve their products. Expect the trend of falling prices and improved features to continue.

The technology for implementing flash disks has matured over the past seven years, and engineers have the option of designing their own solutions or buying products off-the-shelf.

The building-block paradigm has been used extensively to describe the way systems are designed. These blocks range from complete systems to component solutions. For example, multifunction processors are connected to other modules such as display drivers, RF modems, network modules, flash disks (see Photo 1), and so on.

When searching for the appropriate solution, several factors come into play:

- what's the desired time to market?
- is there a ready-made solution available?
- can the dreaded NIH (Not Invented Here) syndrome be overcome?
- what are the costs (both direct and hidden)?

DEFINING REQUIREMENTS

Let's look at some of the requirements you need to consider for a storage system. Will it be used for code, data, or combined storage? Will it be a read-only device, or does it need full read and write capability?

Will the code be executed directly from the device (i.e., execute in place [XIP])? How much storage will be required, what will the performance requirements be, and are there any limits on the form factor?

Next, consider the kind of interface you need. Does the device need to be removable or fixed?

What about memory requirements? Will all the configurations require the same amount of memory? Will storage requirements change in the future?

And of course, there are the marketing questions. How much will it cost? Will the device that gives you the best price and performance today remain the same when you start shipping? And finally, can the product be made in-house?

As well, one of the major factors affecting the final choice is whether or not you're designing the complete system. If the project uses an off-the-shelf SBC, some options are eliminated.

Figure 1—In many cases, design engineers find that the memory address space in PCs is limited. The simplest way to access the flash disk is through a sliding window interface for large flash arrays.

However, the project might use an SBC for prototyping and then migrate to a custom design. In that case, you want to make sure that the same storage solution can be used for both phases.

It's important to understand the differences between using flash memory for direct code execution (e.g., XIP) and as a disk replacement. A modem with firmware stored in flash memory is a good example of an XIP application. This firmware is updated fairly infrequently and executed directly out of flash.

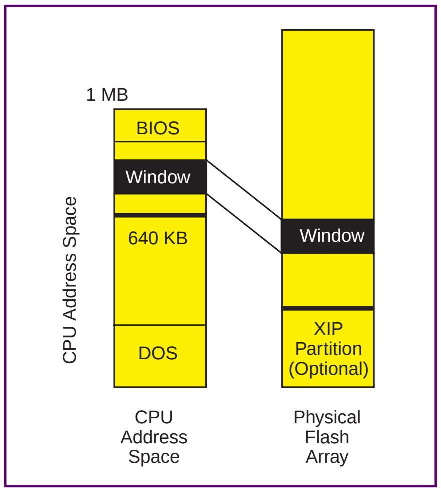
On the other hand, an embedded PC that runs DOS or an RTOS will load programs from the flash disk into memory. These programs are stored as files, and a file system manages data storage and retrieval. On the flash disk, individual files may be updated, whereas in the modem example, usually the entire contents of the flash are updated.

AVAILABLE SOLUTIONS

After you decide to use a flash disk, where do you get the software to make it work? In Raz's article, "An In-Depth Look at FTL," (*INK* 83), he discusses the technology that makes disk emulation using flash memory possible.

FTL has been recognized as the leading technology for linear flash-disk management, and M-Systems' implementation is the industry standard. Therefore, the software part of the project can also use an off-the-shelf solution.

In the following example, we look at a class of solutions called linear flash disks. We won't deal with the more expensive and complex class of solutions based on the ATA or IDE interface. The two classes differ in the type of interface presented to the host.



An ATA flash disk uses a processor to talk to the host system using the ATA protocol. The processor then translates the commands from the host (e.g., read, write, or erase) into operations on the flash devices. In this type of flash disk, there is no direct data path between the processor in the host system and the flash chips in the flash disk.

By contrast, a linear flash disk has a direct data path between the host processor and flash chips. The flash memory is allocated a memory range in the processor's address space. In a linear flash disk, the host processor executes the algorithms that access the flash.

Linear flash disks are inherently lower cost than ATA flash disks because an ATA disk contains an additional processor and has a more complex electrical interface.

TYPICAL PROJECT

Our project has the requirements listed in Table 1. This example uses a PC-type platform, but the design considerations for the flash disk apply to any architecture.

Our goal is to put a device into the hands of shoppers so they can press a trigger and verify the price of an item on the store shelf. In addition, the device will have a display capable of showing graphical advertisements, coupons, or special product announcements.

The unit will keep a log of the items scanned during the day. The price database and advertisement content will be updated and the scan log downloaded every night through the serial interface when the unit is placed in its charging cradle. The storage requirements are 8 MB with the ability to expand to 16 MB or more.

The schedule requires a proof-of-concept model in one month and production

Application	Portable price-verification system for stores
Environment	Mobile, battery operated, used by supermarket shoppers
Form factor	Small, hand-held, sealed unit
Features and user interface	Bar-code scanner, LCD, scan trigger
External interfaces	Serial port, battery charger
Data storage requirements	Application program, price database, graphics for ads
Operating system	DOS or RTOS
Platform	Embedded '386 with a customized PC architecture (diskless, no keyboard)

Table 1—Many portable applications share similar requirements when building their platforms. Here's what you typically need to check when creating a price-verification system.

prototypes in three months. Therefore, the only way to meet the deadlines is to use a commercial SBC for the proof of concept and at the same time design the final unit.

The unit has to be very low cost because every store in a chain needs at least 100 units. Also, it can't have a standard disk drive because of the constraints listed above, yet it needs a disk to make development and maintenance as easy as possible. The solution: a solid-state disk.

The flash-disk solution must allow developers to start prototyping their system with a standard PC and then quickly migrate to the target without modifying the code. The solution would be even better if they could use the same flash disk in the prototype and target systems.

COMPARE THE ALTERNATIVES

Let's analyze two low-cost flash-disk alternatives—a Resident Flash Array (RFA) and a DiskOnChip 2000 from M-Systems—to demonstrate the differences between the make versus buy options.

An RFA consists of flash memory and some control logic soldered directly on the same board as the host processor. A DiskOnChip 2000 contains all the flash and control logic in a small package with a 32-pin DIP interface and is ready to be used in many standard and custom single-board PCs.

BUILDING AN RFA

The PC has limited address space, so a special mechanism is required to access a flash disk, which has several megabytes of storage. The simplest method—a sliding window, as shown in Figure 1—requires a small memory window (typically, 8 KB) in the address space allocated for expansion cards (i.e., segment A000h–EFFFh).

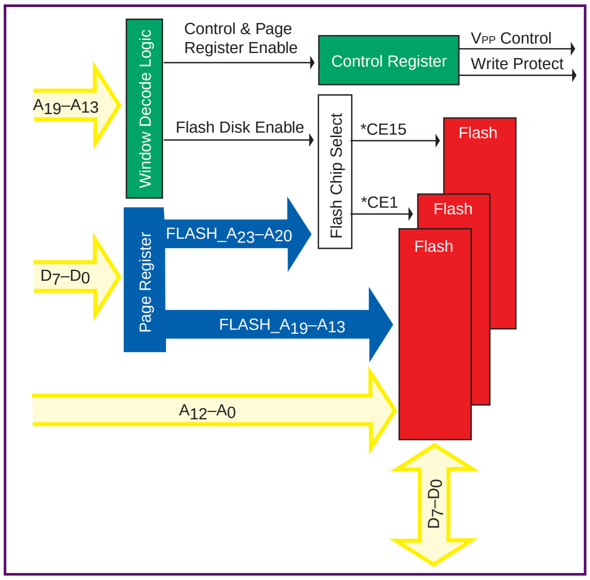


Figure 2—An RFA consists of more than soldering a few flash chips onboard. The function blocks required for implementing a sliding window RFA are flash array, control register, page select register, and the window decode logic.

Figure 2 shows a block diagram of the circuit required to implement an RFA. The base address and window size are determined by the window decode logic.

In this example, we have an 8-KB window into the flash array, so address lines A12–A0 are passed through to the flash chips. The upper address lines, A19–A13, determine the window's base address.

Take care when designing the decode logic to prevent aliasing of this window with memory located above the 1-MB boundary. In addition, DMA and refresh cycles generate addresses within this range, so the appropriate control signals (AEN and REFRESH in PC) must be decoded to prevent erroneous access into the flash array.

The decode logic also determines the location of the Page and Control registers. These registers can be mapped into I/O or memory space, but they must be mapped into a different region or they will conflict with the space used to access the flash chips. Many embedded processors have a built-in chip-select generator circuit that can be used instead of the external decode logic.

Since each flash chip is 1 MB in size and the window is smaller than the chip size, we need a method of driving the upper address. This task is accomplished via the Page register.

The name Page register comes from systems using a paged memory access method. By driving the upper address lines, the register selects a block or a page of flash to be accessed. The number of bits in this register is determined by the size of the flash array and the sliding window.

To access up to 16 MB (1000000h) of flash, we need a total of 24 address lines. The lower 13 address lines are already available, so we need to drive lines A23–A13.

The Page register must be 11 bits wide. If you want to reduce the number of bits in the Page register, you can increase the size of the window. However, a larger window uses up valuable address space.

The PC architecture has limited space for expansion cards. Systems with many expansion cards might have a lot of integration problems, so consider complexity versus ease of integration when designing an interface.

Not all of the address lines generated by the Page register go to the flash chips. The four upper address lines, A23–A20, are routed to a decoder that generates individual chip-select signals for up to 16 flash chips. If a larger array is required, Page register's size can be increased and the additional signals used to encode a larger number of chip selects.

In addition to the logic used to decode the address range, a Control register is usually implemented. This register can be used to control and read the status of V_{PP}, $V_{PPSense}$, write-protect switches, and so on.

Figure 2 doesn't include buffers and transceivers. However, they need to be

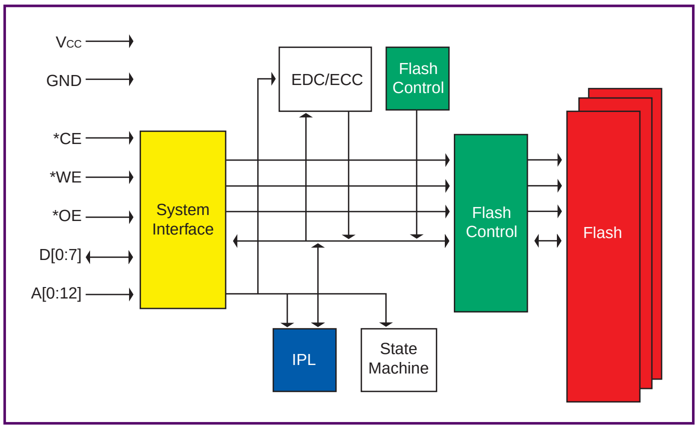


Figure 3—DiskOnChip 2000 provides the design engineer a complete modular solution without having to learn the intricacies of flash. Its major components include flash array, control register, initial program loading (IPL), EDC/ECC, and a common system interface.

added to overcome problems of bus loading and to meet the drive and timing requirements of the entire system.

As you can see, the hardware required to implement an RFA is more than just a handful of flash chips. Several discrete components or programmable logic are required.

This part count—and the board area required to mount the parts—must be taken into account when deciding whether or not to make your own RFA. And once the hardware part of the project is done, you're still faced with the burden of writing the software to make the RFA function as a hard-disk replacement.

DiskOnChip 2000 DESIGN

Working with flash and making it function like a hard disk can be quite complex unless you have a lot of experience or an off-the-shelf package. The most appropriate flash disk for our application is M-System's DiskOnChip 2000.

Illustrated in Figure 3, the DiskOnChip 2000 is a complete flash disk packaged in an industry-standard 32-pin DIP form factor. The interface with the device is quite simple because the device's pinout is similar to an EEPROM or SRAM chip (see Figure 4 and Table 2).

The interface pinout and footprint remain the same regardless of the disk's capacity. Available capacities range from 2 to 72 MB, and future growth options may double or even quadruple the capacity.

The DiskOnChip 2000 shares some of the same system requirements as an RFA. However, unlike designing an RFA, working with the DiskOnChip 2000 is quite simple.

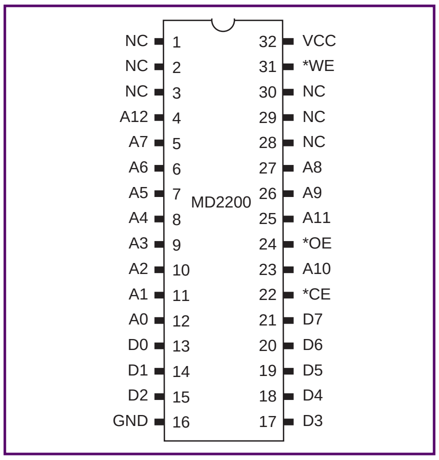


Figure 4—DiskOnChip 2000 has a very simple interface. Since its interface is based on a 32-pin JEDEC standard, the device's pinout is similar to an EEPROM or SRAM chip.

Pin Name	Description	Pin Number	Direction
A0–A12	Address Bus	4–12, 23, 25–27	I
D0–D7	Data Bus	13–15, 17–21	I/O
*CE	Chip Enable	22	I
*OE	Output Enable	24	I
*WE	Write Enable	31	I
NC	Not Connected	1, 2, 3, 28, 29, 30	
VCC	Power	32	
GND	Ground	16	

Table 2—Since the signal definitions of DiskOnChip 2000 are the same pinout as the EEPROM, it can be plugged into an existing socket with minimal or no redesign effort.

This device only requires an 8-KB memory window in the system's address space. It doesn't need a separate I/O address range to access any of the control registers.

All the firmware required to make it function like a hard-disk replacement is already built in as a BIOS extension module. Therefore, it can serve as the boot device in a PC. The DiskOnChip 2000 incorporates an additional feature not available in the RFA—hardware EDC/ECC for maximum data reliability.

The PC architecture has a standard mechanism for adding support for devices that aren't supported by the standard BIOS—a BIOS extension. When properly written, a BIOS extension is executed by the BIOS after the system has completed its Power On Self Test (POST).

At this point, the system can allow other drivers to install themselves. The BIOS scans the expansion memory space between segments C000h and EFFFh and searches for BIOS-extension code. Just like the RFA, the DiskOnChip has to be located at a base address that is within this range, in an area that doesn't conflict with other hardware.

Many SBCs already have a socket that can accommodate the DiskOnChip 2000 with the logic required to generate a chip select in the appropriate address range. If they don't have a socket, there are several options for adding in a DiskOnChip 2000-compatible socket (e.g., via ISA-bus or PC/104 boards).

Feature	DiskOnChip 2000	RFA
Environment	solid-state, low power, withstands high shock and vibration	same
Form Factor	small, modular, <1 in.^{2} regardless of capacity	small, capacity affects real estate (~9 in.^{2} for a 16-MB 28F008 TSOP)
Capacity	2–72 MB now, 144+ MB in the future, expandable after mfg.	limited to amount of flash designed onboard and populated
Flash Technology	interface independent of technology inside device	limited to flash designed onboard
Handling and Preprogramming	preprogrammable with low-cost gang programmer (no special handling)	flash components in a TSOP need special handling to preprogram
Software Support	multiple OSs and RTOSs	requires resource investment to provide same support
Ease of Use	same utilities and function calls access flash disk as a regular hard disk, so third-party tools manage data storage	depends on software solution implemented
Performance	sustained write: >250 kBps sustained read: >700 kBps	depends on technology, with Intel28F008: sustained write: 25 kBps; sustained read: 1 MBps
Prototype and Migrate to Final System	many systems have appropriate socket; if not, ISA-bus and PC/104 cards offer immediate prototyping in a desktop PC and migration migration to target	software development must wait for hardware design, mfg., and testing. No concurrent development of target hardware and application
Price	24 MB in OEM quantities is $174 including support hardware and FFS licenses	approx. $15 per MB of flash, not

Table 3—When deciding which solution to implement in a design, many factors must be considered. This table summarizes the tradeoffs between the DiskOnchip 2000 and an RFA.

The compact package and easy installation make it a modular solution. Storage capacity doesn't need to be predetermined. Boards can be built with a socket and populated just before shipping. Units can be preprogrammed in a gang programmer, while the board is being manufactured.

Software support for this module is much easier than for the RFA. Since it is a standard product, drivers are already available for many operating systems and RTOSs (e.g., DOS, Win3.*x*, Win95, WinNT, WinCE, VxWorks, QNX, pSOS, VRTX, etc.).

In addition, for custom architectures that are not supported, the FLite package is available which provides full source code—all in ANSI C—for the drivers.

ANALYZING THE SOLUTIONS

Now that we've reviewed the implementation requirements for the two flash-disk options, let's look back at the requirements we defined and determine the best solution.

Based on the results summarized in Table 3, there is little motivation to develop our own flash-disk solution. Especially if time to market is critical and you don't have a lot of experience in working with flash chips, a proven solution might mean the difference between getting your product out to market and not getting it out at all.

DiskOnChip 2000 lets you treat the storage subsystem as a building block. So, you can spend your engineering resources on addressing issues you know how to deal with the best—your application. EPC

Raz Dan is the customer engineering manager for M-Systems. He is currently responsible for custom applications, advanced technical support, and system integration for the company's product line. Raz holds a BSEE from Tel-Aviv University. You may reach him at raz@ccm.msyscal.com.

Stefanie L. Hein has worked in the electronics industry for the past four years, after receiving her BS and BA from Brigham Young University. She is currently the corporate communications manager for M-Systems.

SOURCE
DiskOnChip 2000, PC Flash Disks and Cards, Fast Flash Disk (FFD), FLite
M-Systems, Inc.
39899 Balentine Dr., Ste. 335
Newark, CA 94560
(510) 413-5950
Fax: (510) 413-5980
info@ccm.msyscal.com
www.m-sys.com

IRS

416 Very Useful
417 Moderately Useful
418 Not Useful

Marc Guillemont

Real-Time Operating Systems

Part 1: Fundamental Components

The embedded PC is a good match for the power, resources, cost, and complexity of the real-time market. Marc launches this series on real-time operating systems with simple answers to basic questions like "What makes a real-time OS?"

Your body is much akin to a real-time operating system. Your brain contains the kernel executing the threads that determine your everyday life. You can communicate with other units, and you are time aware. You can be interrupted, and you can be scheduled.

You have the ability to manage your memory, monitor your actions, and correct your mistakes. In a manner of speaking, you're a real-time device running a real-time operating system (RTOS).

In a process-based real-time environment, system components (processes) are kept from affecting each other through memory protection (normally via an MMU). Bugs such as stray pointers or array indices may corrupt other processes if this mechanism isn't available. The MMU can also identify the violation to the RTOS, which then takes appropriate action, retaining intelligent control over the system.

Processes communicate via a well-defined interprocess communication (IPC) interface. In the body, these components are things like muscles, which your brain communicates with via a nerve IPC. In some cases, it may make sense to group control of related muscles through threads.

Protection between components prevents something like a broken finger from directly affecting your digestion. The body's MMU identifies the break to the brain (i.e., RTOS), which uses IPC to tell the surrounding tissue to start repairing the break, intelligently recovering from the fault.

RTOSs can be found in a variety of places other than the human body. PBXs, smart home devices, medical equipment, automobiles, and scientific data-gathering devices are a few that come to mind.

With this proliferation of embedded real-time activity, I want to introduce you to the world of RTOSs from the inside out. So, I'll discuss the modules and techniques used to implement a typical RTOS.

WHY USE AN RTOS?

Most applications don't need to run in a real-time environment. Device programmer applications are one good example. The application focuses on programming a single device at a time. No other real-world events may interrupt the programming process.

Thus, no memory management, scheduling, or monitoring needs to take place. The only real requirement is that some sort of executive is running to provide the necessary hardware and software resources for the application.

On the other hand, an Internet appliance like a remotely monitored weather station needs a real-time operating environment. And, some systems can run in so-called soft real time (see the "Soft Real Time" sidebar).

Since in a real-time system, multiple operations occur asynchronously and at predetermined times, multiple software threads have to be capable of running concurrently, as shown in Figure 1.

This concurrent operation forces the application programmer to be concerned with synchronization and memory management as well as interrupt handling, timing, and intermodule and intersite communications. The RTOS provides the means by which all these processes are handled from within the framework of the RTOS code.

Think of an RTOS as an engine running various applications. If you view an RTOS application suite as an automobile, the RTOS is the motor, and the application is the frame and body of the car.

In that context, the motor or RTOS could power many types of automobiles or applications. That is, a particular RTOS can serve just as well in a medical application as in an automotive application.

The only difference between most RTOS implementations is the coding and purpose of the application being supported. Let's take an RTOS apart module by module and examine how each piece fits into the grand scheme.

Soft Real Time—Greg Bergsma, QNX Software Systems

A good example of a soft real-time system would be an ATM. Ideally the machine should process your request in less than a few seconds, but if you had to, you'd probably wait a minute for the response. If it takes any longer, you're likely to look for another machine or go to a branch office.

The stress here is more on throughput when under load. A late response may be acceptable, but it's less than ideal. Soft real-time systems can tolerate the occasional missed deadline, but they still must operate very quickly and repeatably.

RTOS MODULARITY

To be flexible and efficient, an RTOS should be modular (see Figure 2). If the real-time application is small or simplistic, a nonmodular single OS kernel may be sufficient. But if it must span various platform sizes and encompass differing sets of functionality, a modular RTOS is in order.

In a modular system, OS services such as scheduling, memory management, communications, and timing are provided as separate program modules. The selection and inclusion of certain modules form the basis for the RTOS's functionality as it relates to the kernel.

For modularity to work, the mechanism by which a module works must be separate from the policy it invokes. The kernel must provide the basic framework so supporting modules can implement differing processes without compromising the kernel's inherent mechanisms.

This situation calls for an interface that can support the smallest to the largest application configurations, as well as the simplest to most sophisticated modules.

The concept of modularity implies that individual processes are combined in the most efficient manner to drive a particular application. In most instances, RTOS modularity is implemented at RTOS build time.

High availability is achieved since modules can be replaced, repaired, or removed on-the-fly. Although this on-the-fly ability is desirable in some situations (e.g., candy making and banking), it's not necessary in all RTOS applications. For instance, you probably wouldn't replace software modules in a washing-machine application.

Modularity enables the programmer to implement new modules that complement the standard kernel procedures. The new modules may introduce new procedures or custom implementations of the standard kernel interface.

Including modularity in an RTOS facilitates unlimited customization of the kernel to satisfy specific functionality, size, or hardware support requirements. As an added advantage, unwanted processes need not be included, thus releasing resources for other components of the RTOS application.

THE KERNEL

An RTOS is a collection of software modules that cooperate, enabling computational hardware and application code to perform real work in real time. The heart of this collaboration is the kernel.

The kernel provides the basis for RTOS operation. Normally, it exports services via an API to manage system resources and application processes (e.g., synchronization, interrupt handling, trap, and exception handling).

Kernel services are accessed by other modules via the kernel interface. This interface isn't exported directly to any application. It's dedicated to system components, and it defines exported functions to other modules and the operation of the functions expected from them.

The kernel can also be responsible for initializing the target processor and setting up low-level details of the application (boot activity). So, it's safe to say the kernel is the low-level foundation of all RTOSs.

RTOS INTERRUPTS

Once the kernel has prepared the system for operation, the application in association with the kernel-provided services is

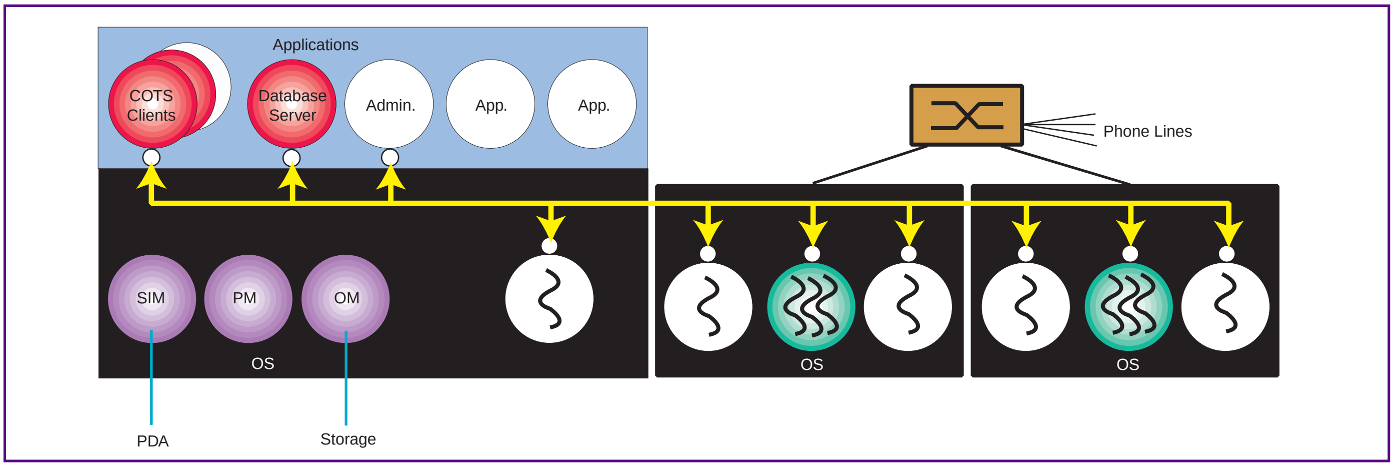


Figure 1—Once a single system is built, it can be deployed throughout a set of boards and in different configurations. Since the IPC is location transparent, the deployed configuration can be dynamically changed according to changing needs.

ready to handle software- or hardware-generated events. In nonreal-time environments, a process called polling does this.

Polling involves interrogating ports, software modules, and devices one at a time in a predetermined order. Polling implies that the device that gets service is the device being polled when it requires service.

The drawback with polling is that other devices can need service but must wait until the currently selected device is fully serviced.

In the real-time world, devices and software modules can interrupt the process and request service. This scheme gives the real-time application the ability to prioritize service requests. The kernel or an external attachable module can contain the interrupt-management routines.

THREADS

Event-driven real-time systems usually run a monitor routine while waiting for an event to trigger an action.

In a real-time system, the other active programmatical entity is usually a thread. A thread is a unit of execution representing a single flow of sequential execution of a program (see Figure 3).

Threads can be application or kernel based. Application-based threads execute in nonprivileged mode usually in an isolated address space. Kernel-based threads execute in privileged processor mode with access to restricted processor instructions in system address space.

Under program control, it's possible for an application thread to operate in kernel-based mode. This situation usually results from a trap or exception condition.

A thread is created in its host application. During its life, a thread may cross its application boundaries and execute code belonging to another application. This situation enables efficient interapplication cooperations, although they're usually restricted to threads running in the system address space.

Conversely, a thread may be executed on behalf of another thread (local or remote) that's requesting a given service which the thread is able to provide. This flexibility in the thread's identity is another key for efficiency in modular systems.

In a process-based model, by contrast, multiple threads running in a process address space are memory protected from threads running in another process address space.

SCHEDULING THREADS

Because threads are units of execution within applications, it's logical to assume that in a real-time situation, threads must be able to run concurrently or be preempted by higher priority processes.

The kernel exports an interface to enable the programmer to use any scheduler module. Also, if needed, it lets the system programmer introduce new scheduling modules implementing particular scheduling policies. For now, let's think of a scheduler as modular with the ability to interface to an exported kernel-scheduler interface.

As you have ascertained, RTOSs require a scheduler to operate. A scheduler is a module that provides scheduling rules and regulations. A scheduling decision may be based on thread priority, response times, or any other time-critical criteria affecting the operation of the real-time system.

The exported kernel interface allows coordination of thread execution. Thread execution with other threads, events, and functions is controlled by the scheduler module, which lets a thread wait until an

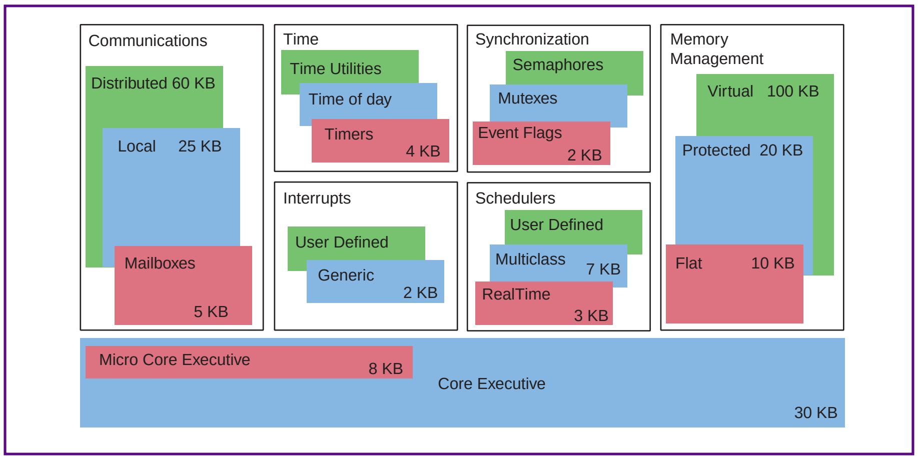


Figure 2—A componentized RTOS (like Sun Microsystem's ChorusOS) shifts the focus from microkernels to components. The whole design stresses separation of mechanisms from policies. So, scheduling, interrupt and fault management, memory management, and IPC are provided by external modules and are not implemented within the base of the system (i.e., the Core Executive).

event is posted to it, stop or restart another thread, stop or restart all threads in any running application, and abort other threads.

RTOS MEMORY MANAGER

So far, I've described a typical RTOS from the ground up. I first went over the importance of RTOS modularity and kernel concepts and then moved on to interrupt handling and thread scheduling.

These pieces can be combined to assemble a minimal real-time system. But to bring them to life as a working RTOS, we need another ingredient—memory.

As you well know, most of the time, memory is physical hardware. In a working RTOS, it can be physical silicon (i.e., the processor issues addresses which directly represent cells of physical memory).

But more often, the processor issues addresses that are translated (by hardware) to determine the physical cell concerned. This hardware mechanism—the MMU (Memory Management Unit)—brings many advantages, the most obvious being a hardware-based protection between different programs. It dramatically increases the system's reliability at virtually no execution cost overhead.

At least two memory models must be taken into consideration when working with an RTOS—the flat model and the protected model.

The flat model needs the least management—if any at all—because all applications are run in one unprotected address space. All virtual addresses directly map to their respective physical address, and an application may allocate no more memory than is physically available.

The protected memory model is the entry-level memory-management scheme for real-time applications. It exploits the machine's hardware capabilities to offer separate protected address spaces for different applications. This model also eases the mapping of various special memory locations (e.g., video RAM) in an application's address space.

Virtual memory mode is another advanced memory-management technique that uses physical storage to emulate real memory. This mode uses paging and swapping to provide the appearance of more real memory than physically exists within the RTOS.

SYNCHRONIZATION

RTOSs are event-driven mechanisms. Since events can occur anytime during any process, there must be a logical way to coordinate incoming events. The synchronization module performs this work.

The synchronization module provides its services via sychronization objects such as an integer counter associated with a thread's waiting queue. On initialization, the counter is loaded with a user-defined positive or null value. These queue counters are located within the application address space and decremented when a thread performs a wait operation.

If the count goes negative, the thread is blocked and put in the respective queue. If the decrement operation does not take the counter negative, the thread continues normal execution.

Another type of synchronization module supplies sleep locks. This type of synchronization object is an exclusion lock where a thread sleeps instead of spinning when contention occurs.

Earlier, I mentioned posting events to threads. The event module contains the algorithms necessary to effect posting. A set of bits called the event-flags set resides in memory and is associated with a thread wait queue.

Each bit represents one event. An event is posted when a flag bit is set. This flag bit can be used by interrupt handlers and threads for signaling purposes.

In a process-based model, IPC can serve as a synchronization mechanism.

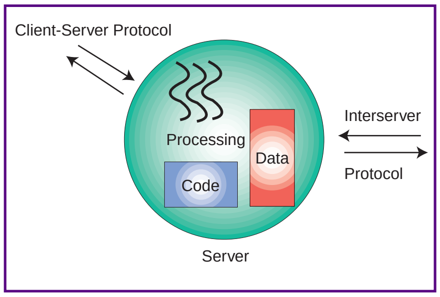


Figure 3—The basic building block of ChorusOS applications is the actor that represents the basic unit of resource allocation and is the shell into which threads can execute. The actor also represents the basic error-detection (exception handlers are attached to actors) and fault-confinement (protected address space and execution boundary are associated to actors) abstraction.

TIME MANAGEMENT

It's good to be able to schedule and synchronize events and responses to events within a real-time application, but execution would be chaotic without an underlying time base. Timing for an RTOS and its associated applications is provided by a set of time-management routines.

RTOS time management generally focuses on time-out management and watchdog events. Date and time may be present as well. Performance-measurement and time-interval functionality play an integral role.

For example, imagine that one RTOS-controlled electronic element needs to synchronize its operation with a remote electronic device. The timing accuracy between the remote devices is then governed by the time-management modules.

Event recovery is also important in most RTOS-based systems. The watchdog timer is usually employed for this task.

COMMUNICATIONS

Each process running under RTOS control is designed to perform a specific task. Some tasks may depend on the state of another task inside or outside the local RTOS environment.

A communications conduit must be in place for all subtasks and applications to communicate with each other. The communications module effects this, and this process is known as IPC.

This module enables a major application composed of multiple threads to send and receive chunks of data (i.e., messages) between threads and applications, inside or outside the same local RTOS.

One model for IPC implementation is to create message pools, queues, or mailboxes shared by all or a subset of threads and applications. Messages are stored in these pools, and pointers are passed between applications to retrieve messages attached to the pools.

This communication scheme is the most efficient, as information is never copied but stored directly where it will be retrieved.

To effect this messaging technique over a network, a logical mailbox system is used, similar to a basic E-mail system. Messages are exchanged between logical addresses, often known as ports.

In some cases, this network-wide IPC totally hides the distributed nature of the application. Local and remote communications are

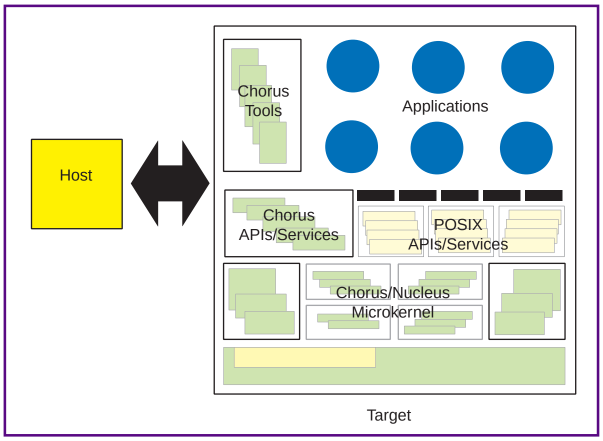


Figure 4—When building an RTOS for a specific application, you use a host platform like Windows NT or Unix to design the applications that run on the RTOS on the target system. This diagram represents the development configuration for developing on a ChorusOS operating system.

expressed in the same way, which simplifies the IPC's programming, eases different distributions of its components, and permits easier static or dynamic configuration and reconfiguration.

In other words, this system benefits from the underlying hardware redundancy and flexibility. Of course, the built-in RTOS message synchronization controls the flow of reading and writing messages.

MONITORING AND DEBUGGING

Given the complexity of RTOS environments, so you may need to monitor activity within the kernel and application modules (see Figure 4). This task is most often done by a monitor module running concurrently with the RTOS and application.

Major events are logged with pertinent process information that tells the programmer what occurred when the event was recognized and processed. As a result, the process or application can be controlled and possibly adapted.

Once the desired data is captured, the programmer can recall the statistics and determine the application code's performance. For instance, for tuning purposes, it's necessary to know how much compute time is spent in certain routines. With this information, the programmer can tell where the program consumes the most resources.

Using a debugger, the programmer can single-step through an application or run the application until a failure occurs. At that point, a memory dump can be taken for analysis of the problem.

To take this concept one step further, hardware debugging tools can assist the software-based debugger by capturing and logging the resultant hardware reactions as they relate to the application code. Hardware debuggers are minimally intrusive and usually provide a means by which the debug pass can be recorded and replayed.

REAL-TIME WORLD

In this article, I examined the major components of a typical RTOS. As you see, there are endless variations that may be applied to the many modules of an RTOS.

What you should walk away with is that RTOSs are time-dependent, event-driven environments suitable for supporting real-life, real-time applications.

In Part 2, I'll take the logical framework of an RTOS and interface it to hardware in the real world. RPC.EPC

Marc Guillemont, ChorusOS product manager for Sun Microsystem's Embedded Systems Group, joined INRIA in 1977 to work on the Cyclades project. He was a member of the initial Chorus research project team in 1980 before becoming head of the team. Marc managed the final research phases of ChorusOS before developing the commercial version. You may reach him at marc.guillemont@france.sun.com.

SOURCE
ChorusOS
Sun Microsystems
Embedded Systems Software Group
1999 S. Bascom Ave., Ste. 400
Campbell, CA 95008
(408) 879-4100
Fax: (408) 879-4102
www.sun.com/chorus

IRS

419 Very Useful
420 Moderately Useful
421 Not Useful

Fred Eady

RF Telemetry

Part 1: Theory and Implementation

Although RF telemetry is traditionally associated with rocket science, it's as down to earth as modem-based communications. Using a Linx UHF receiver module, Fred pokes at low-power data transmission. After all, who wants to alert the FCC?

I don't know about you, but when I hear "telemetry," I immediately go to the stars. I'll bet you equate telemetry and satellites, too. It's natural in today's society.

For me to be spouting about telemetry can only mean one thing. I'm bringing that concept down to Earth, and embedding it.

My first real experience with any kind of telemetry was with model rocketry. I must have fired off a thousand of those things with almost every payload I could think of.

The problem was, in those days, you had to build everything from scratch. Consequently, any electronic payloads in my younger days were very amateurish.

While doing research for this article, I discovered things haven't changed much for the model-rocket scientist. Very few vendors cater to selling radios with rockets.

Enough reminiscing. I want to talk about what telemetry is and why we should bother with it.

TELEMETRY 101

If I had to sum it up, I'd say telemetry is a means by which data is collected from a remote-monitoring device over a predetermined communications method. If you don't work in the telemetry field every day, the most common form of telemetry you may be familiar with is satellite oriented.

Although the concept is fairly simple, the implementation isn't. First, you design and build your satellite. Then, you have to get it up there somehow. Assuming all goes well, you need a pretty elaborate base station to receive and assimilate your data.

The most common form of satellite-based telemetry the average person encounters is the six o'clock news weather report. About the closest I'll ever get to a telemetry satellite is stepping out my back door and watching a launch from the Cape Canaveral Air Force Station.

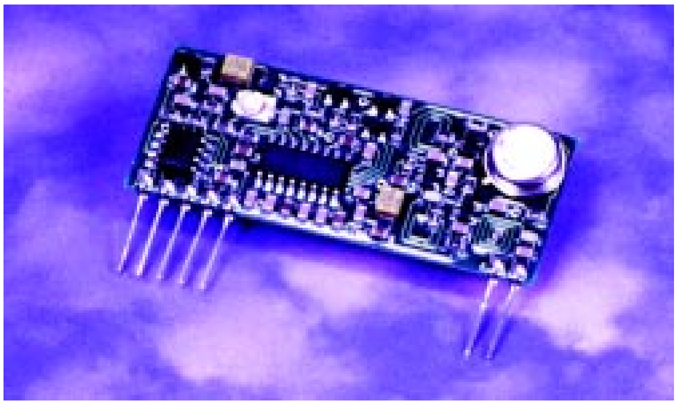


Photo 1—This is something only an RF engineer could love—ground planes and coils. I think I'll stick with my buses and ports, thank you.

If you need to get close, companies like Handar can take you there. Handar makes a wide range of products that are GOES-satellite compatible.

If satellites are a little bit much, there's RF modem-based telemetry. This type of telemetry involves moving data across terrain and boundaries that prove difficult for traditional land-based communications channels.

For instance, suppose you, a member of a research team, are in the rain forest tracking a specific species of monkey. Problem. The monkeys are across the river from your base camp.

It's your turn to hike while your colleagues monitor equipment in a tent at base camp. Since you're working in dense foliage, you can't bounce your signal off a satellite even if you had access to one. Running wires across the river isn't even a consideration.

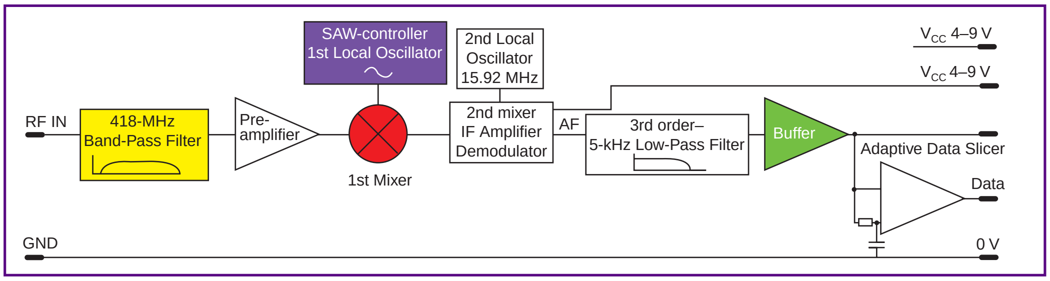


Figure 1—It's surely smoke and mirrors. How can all this stuff be on that little board? Note the data slicer in the bottom right corner.

Infrared or laser technology? Nope. Not here. That's line of sight, and you can't see the forest for the trees.

Aha! Microwaves! Naaah. First good rain would wash out your signal.

What's left? Good old RF.

Unlike satellite telemetry, RF modem-based telemetry is less complicated to use. Essentially, you have a transmitter-receiver pair capable of translating sensor data into modulated RF energy and vice versa.

Depending on the application, the RF modem pair's output power can vary from milliwatts to multiwatts. Normally, you see RF modem technology employed at plant sites or in the field as portable weather stations or monitoring sites.

We could talk about this topic for days. But, let's get on with it and lay the groundwork for putting embedded hardware on the air.

RF MAGIC

The first problem I pondered was how I was going to build up a working RF modem. Yep, I've got the big fancy FCC license. So what? I'm a computer dude, not a RF engineer.

Speaking of the FCC, I don't want to have to get a special license to use this thing. Besides, unlike my little 1 and 0 world, RF is smoke and mirrors.

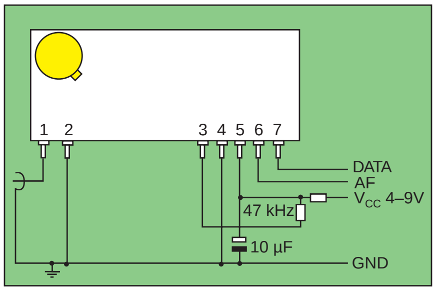


Figure 2—This is the base test circuit. Carrier detect is pulled up when it's not being used.

I decided to check out the availability of a ready-to-roll RF modem that I could easily tie to a couple pieces of embedded hardware and fly binary digits across the ether. As luck would have it, my first possibility was an out-of-country solution. It was perfect!

An RF modem with enough power for me to put some distance between the embedded PCs was simple to implement. Well, not really.

After talking to the engineer, I found my "perfect" solution was not FCC approved. But, if I wanted to use it in the outback, there would be no problem. Next!

With that tidbit of knowledge, I decided to make sure I searched within the boundaries of the US and the regulations of the FCC. Restrictions such as power output and licensing requirements narrowed the field.

I needed a low-power solution that required no FCC approval on my part and was easy to integrate with my embedded platforms. It would be nice, too, if I found a solution small enough and power-stingy enough to fit in the nose of a model rocket.

Well, ask and ye shall receive. Linx Technologies—wireless made simple. The transmitter and receiver pair is modular and easily integrated into most application. As long as I don't need to traverse super-long distances, the Linx solution is adequate.

RF 101

"Wireless made simple" did it for me. In my AM radio days (a few years back), I calculated power output and antenna patterns with ease. I have no desire to retool that talent. I passed the FCC exam and qualified to operate the transmitter. That's all I needed (or wanted) to do.

I didn't own a computer (PCs as we know them weren't commercially available), and I wouldn't have known what to do with one if I did. The closest I got to RF telemetry as a kid was hang a light-sensitive audio oscillator off my walkie-talkie.

Times have changed. I have a computer or two—embedded even. And, I have a data-ready RF modem set. I'm dangerous.

I considered the Linx RXM-418 UHF receiver module to capture my telemetry data. This module incorporates an ultra-sensitive, SAW-based, double-conversion FM superheterodyne receiver. When paired with the Linx TXM-418 transmitter module, the units create a highly reliable RF link capable of transferring analog or digital data at distances in excess of 500′.

THE RECEIVING END

As I said, the RXM-418 is a SAW (Surface Acoustic Wave) based double-conversion FM superheterodyne receiver. The receiver uses a data slicer (it slices, it dices) that is driven by the AF output.

From what I can see, the data slicer is effectively a Schmitt trigger. A depiction of the RXM-418 is rendered in Figure 1.

You all know I like to blink the lights, and there's a carrier-detect signal available to indicate to external circuits that a signal is present. This signal could be used in a power-saver circuit or to indicate to an embedded PC that a signal is being received. All good RF stuff. All I have to do is add an antenna.

The range of the RF link depends on many factors, including the type of antenna employed and the environment. According to the datasheets, the 500′ range is a conservative estimate of the operating distance over open ground using ¼-whip antennae at both ends of the link at 5′ above ground.

A smaller antenna, obstacles, or interference can reduce the reliable operating range down to as little as 100′. By raising the antenna higher than 5′, slowing the data rate, or using a larger antenna on the receiver, ranges in excess of 1500′ can be realized.

The Linx folks don't recommend you toy with the FCC regulations by doing this. They offer a service to keep you out of such trouble.

As Photo 1 shows, the receiver is packaged as a hybrid SIP module with five pins spaced on 0.100″ centers. This packaging enables easy horizontal and vertical mounting in tight spaces. The standard-spaced leads make it easy to hook this little wonder into almost any prototype circuitry.

The RXM-418 is tough, too. The module is coated with Tek-Protek encapsulant to provide some shock resistance and prevent damage that can be dealt from operating in hostile environments.

Reference Figure 2 as I take a quick turn around the RXM-418. The receiver antenna connects to pin 1 and is capacitively isolated from the internal circuit. Pin 2 should be connected to the RF ground plane and is internally connected to pin 4, which is the power-supply ground.

Pin 3 provides the carrier-detect signal and is designed to directly drive the base of a PNP small-signal transistor. The RXM-418 can operate with clean and stable supply voltages of 4–9 V and only draws a meager 13 mA.

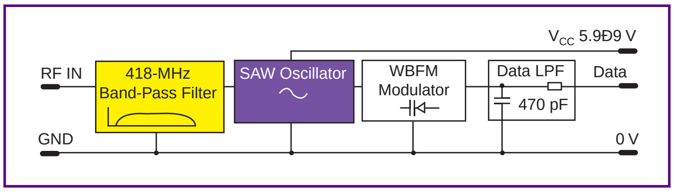


Figure 3—The transmitter is a bit simpler, but it's still smoke and mirrors to me.

Analog data emanates from the FM demodulator output (pin 6). This output is capable of driving analog data circuitry such as modem chips and DTMF decoders.

The digital output from the internal data slicer is a squared version of the signal on the AF output. The output is an exact recreation of the input placed on the transmitter's data pin. If the signal is digital, that's what our embedded PC wants to see.

THE SENDER

Figure 3 shows us that the transmitter is a relatively simple device with respect to the receiver. The SAW technology in both the receiver and transmitter inherently provides good noise margins because the operating frequency stability is very tight.

Just in case you haven't figured it out yet, the RXM-418 and TXM-418 operate at what frequency? I'll give you five guesses, and the first four don't count.

ENCODING THE DATA

Now that we're all qualified and certified RF engineers and can generate electromagnetic fluctuations, how in the world are we going to encode the data we must send? The good news is that due to the features included in the design of our Linx modules, we can encode data in a number of ways.

One of my favorite encoding schemes for analog data is pulse width modulation (PWM). Using some homegrown software and almost any embedded PC, I can translate analog-based signals into a digital form that can be spewed out across the airwaves.

The concept is relatively simple. A quick look at the datasheets for the RF-modem equipment puts the minimum pulse width that can be propagated and received at 10 µs. Maximum rated throughput is stated to be 5 kbps.

Arbitrarily, let's select 1 ms for the pulse width of a binary 0 and 2 ms for a binary 1. With an ADC, the embedded hardware running the binary-to-pulse-width program can encode the digital output and port it out to the input of the RF transmitter. Conversely, on the receiving end, the pulses are received and converted to their digital equivalents. What if there was analog data that could be sent without first having to convert it to digital form? A good example of such analog data is standard touch-tones or DTMF.

As you know, DTMF tones consist of a combination of frequencies that are commonly used for electronic signaling within the public telephone system. You'll find a bunch of DTMF applications in the amateur-radio world, too. Since there are a variety of commercial ICs specifically designed to manipulate DTMF, it's a popular way to convey information or control remotely located devices.

By design, my RF modem can process raw analog signals on both the transmit and receive ends of the RF link. Thus, there's another way to implement an RF telemetry link.

Of course, the most straightforward telemetry link would be direct input of digital data from the serial or parallel port of an embedded PC. If no sensors were used, the embedded device would likely work in a supervisory control mode. This mode would be useful in cases where remote sensors needed periodic polling from the base station to obtain the required telemetry data.

GOTCHA!

So, you think everything's just rosy. The world is finally in harmony with your inner spirits. Life is good.

Sorry! Although everything I've discussed makes RF modem life easy, it's all illegal!

Yep, illegal. Bogus. Part 15 of the FCC regulations doesn't allow "data" to be transferred on the devices I described. "Data"

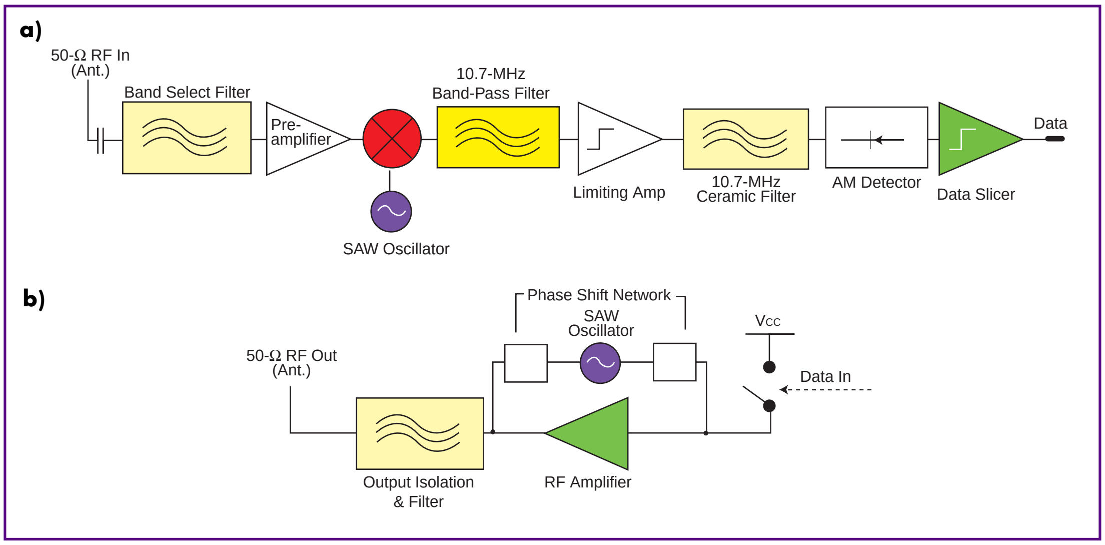


Figure 4—For the RF challenged, the transmitter is the one that says "RF Out."

is defined as collected data such as temperature or humidity. Only "commands" or control signals are legal.

Essentially, anything originating from a computer is considered data. It looks like all is lost, but there's a provision for data transmission under Part 15 paragraph E.

First, cut the output power in half, and then limit your data transmissions to 1 s with a duty cycle of 30 to 1 off versus on time. OK. Now you can transmit "data." Whoopee.

And, oh yeah, remember that the RXM and TXM modules are FM based. So, all the nasties that go with FM transmission and reception are present and accounted for.

A BETTER WAY

Just like the TXM and RXM modules, the Linx LC series is designed to make RFing a pleasure. The LC-TXM and RXM are also SAW based. This variant uses a CPCA (Carrier-Present Carrier-Absent) transmitter capable of sending serial data at up to 5 kbps.

Mixing the magic number of 1s and 0s, the CPCA modulation method enables the transmitter to be legally operated at higher output power levels than conventional continuous carrier modulation methods. The LC series has a shorter range of 300′ line of sight.

To combat the FM-inherent shortcomings, the LC series uses CPCA modulation. This AM-modulation scheme simply suppresses the carrier when the data input is low.

CPCA AM modulation is often referred to by other designations, including CW and OOK (On-Off-Keying). A logic-low 0 is represented by the absence of a carrier, and a logic-high 1 by the presence of a carrier.

During this carrier-deficient period, the part draws little to no power and emits zero signal into the ether. Because FCC regulations measure power output as a function of time, up to twice the output power can be radiated if the data input duty cycle is 50%.

One drawback of this modulation method is that no raw analog or clocked data can be exchanged between the transmitter and receiver. A combined logical view of the LC transmitter and receiver resides within Figure 4.

A WALK AROUND THE TOWER

For most of you, this may have been a different experience. In *EPC*, you're used to reading about hardware speeds and software feeds of various embedded systems. But, I wanted to expose you to the reality of designing RF-based embedded applications.

Just because Part 15 of the FCC regulations implies "unlicensed" doesn't mean you can just go compute and radiate at will. Most of my conversations with RF engineers were laced with warnings about what I could and could not transmit.

One of the first questions I was asked was, "Do you have a ham ticket?" Their point—if I was going to experiment with this stuff, I should do it in the amateur-radio band and save myself the hassle.

Implementing an off-the-shelf embedded RF-telemetry solution isn't simple. To further complicate matters, the designer must design or choose the proper antenna system.

Again, I could write a whole series of articles on that. And yes, that's FCC regulated, too! This area can make or break your RF design, so the folks at Linx told me that a really good app note on antennae will be available on their Web site soon.

This has been a good, bad, and ugly description of one particular set of RF modules that could be used to overcome the natural and FCC obstacles that stand in the way of implementing a reliable RF data link.

Although the modules I described are capable, they might not meet your application's needs. There are many other RF solutions out there. I'll dig them out for you.

In the next installment, I'll take a look at how to get the RF side to cooperate with the embedded side and put some bits into the wind. APC.EPC

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

SOURCES
RXM-418, TXM-418, LC series
Linx Technologies, Inc.
575 S.E. Ashley Pl.
Grants Pass, OR 97526
(541) 471-6256
Fax: (541) 471-6251
www.linxtechnologies.com

Abacom Technologies
67 Hamptonbrook Dr.
Etoblcoke, ON
Canada M9P 1A2
(416) 242-3120
Fax: 242-2697
abacomtech@compuserve.com

IRS

422 Very Useful
423 Moderately Useful
424 Not Useful

MICRO SERIES

Applying Direct Digital Synthesis

Tom Napier

Building and Testing an NCO Generator

Part 2 of 2

Last month, Tom went over the basics of how to synthesize accurate sine- and square-wave signals. This time, he shows you how to build and test a generator with both sine- and square-wave outputs.

In Part 1, I introduced the Numerically Controlled Oscillator (NCO), a chip that can generate a sine wave whose frequency is proportional to a number supplied by the user.

The NCO contains an accumulator to which it continuously adds the user's number at a crystal-controlled rate. So, the accumulator overflows at a frequency that depends only on the length of the accumulator, the crystal frequency, and the increment value.

The NCO chip also converts the ramp output from the accumulator into a series of samples that closely approximates a sine wave at the desired frequency. An external DAC and a low-pass filter convert these samples into a pure sine wave. The sine-wave frequency can be set in millihertz steps from 0 up to 25 MHz (typically).

Using a Harris HSP45102 serial input NCO and a PIC16C54 microcontroller, I designed and built a bench-top signal generator with a frequency range of 1 Hz to 9.999 MHz.

This generator supplies sine- and square-wave outputs. Either Frequency Shift Keyed (FSK) or Binary Phase Shift Keyed (BPSK) modulation can be used.

This month, I cover the details of the design, show how you can build and test your own versatile signal generator, and give examples of the intriguing signals it can generate.

CONTROLLING THE NCO

Apart from the 12 output and two phase-control bits mentioned in Part 1, the NCO chip has six bits that control the loading of a new frequency and two internal control signals that need to be in the correct state before an output can be generated.

The serial input data is controlled by a data-input pin (13), clock pin (14), and serial-enable pin (10). Since the PIC generates a burst of 64 clock pulses as the 64 data bits are loaded, there's no need for a separate enable signal. I wired pin 10 low.

I wired the input-direction pin (11) high since the data is loaded most significant bit first. When pin 17 is low, new frequency data transfers to the increment register. Pin 9 selects which of the two frequencies to transfer.

Since I wanted pin 9 for frequency modulation, the controller keeps pin 17 low except when loading a new pair of frequencies. The increment register is transparent when pin 17 is low, so switching pin 9 changes the frequency instantly.

When pin 12 is high, it stops the accumulator from changing. This situation can stop the output and restart it from the same phase. I wired it to a pin on the PIC, but I don't use it.

Pin 18 disconnects the accumulator feedback. When it's low, the accumulator content is made equal to the increment number. So, any fixed number can be sent to the DAC, and it's useful for testing. I use this feature to switch between plus and minus full scale to adjust the output amplitude.

SQUARE-WAVE OUTPUT

Instead, the square-wave output comes from a comparator chip driven by the filtered sine wave. I used an Analog Devices AD9696KN, which costs about $5 and has rise and fall times less than 5 ns.

I had to wire it with hysteresis (positive feedback), or its low-frequency output would contain a burst of clock frequency every time the sine wave passed through zero. (If you put a scope probe on the comparator outputs, you'll see what I mean. Luckily, the output is stable when no probe is connected.)

Since the square-wave frequency is accurately known and contains odd harmonics up to at least 200 MHz, this output can be a useful calibration source for receiver testing. The square-wave output can also be phase and frequency modulated.

FILTER TEST

A filter test is built into the firmware but isn't accessible from the front panel. If the two outer pins of the mode switch are connected together (e.g., with a clip-on jumper wire) and the switch is not in its center position, the NCO generates a full-scale square wave at ~250 kHz. This wave calibrates the signal amplitude and checks filter performance.

As Figures 1 and 2 show, the major components of the NCO generator are the thumb-wheel switch block, PIC controller, switch decoder, modulation buffer and selector, NCO and DAC chips, the filter and its buffer, and the comparator. It also has a regulated +5-/–5.2-V power supply.

The PIC drives a 74HCT138 eight-way decoder chip to address each of the five four-bit thumb-wheel switches. The range switch has only four states, so only its lower two bits are needed. The two outer pins of the mode switch are wired as if they were the upper two range switch bits.

Diodes connected to the switches let the switch bits be wire-ORed into four input bits of the PIC. Two spare decoder states generate the serial data for the NCO, since the 18-pin PIC doesn't have enough outputs to drive all the NCO functions.

Half of a 74ACT74 dual flip-flop divides the crystal frequency in two to drive the PIC. The other half buffers the applied modulation to synchronize it to the NCO's operation. A 74ACT153 multiplexer switches the modulation to the NCO.

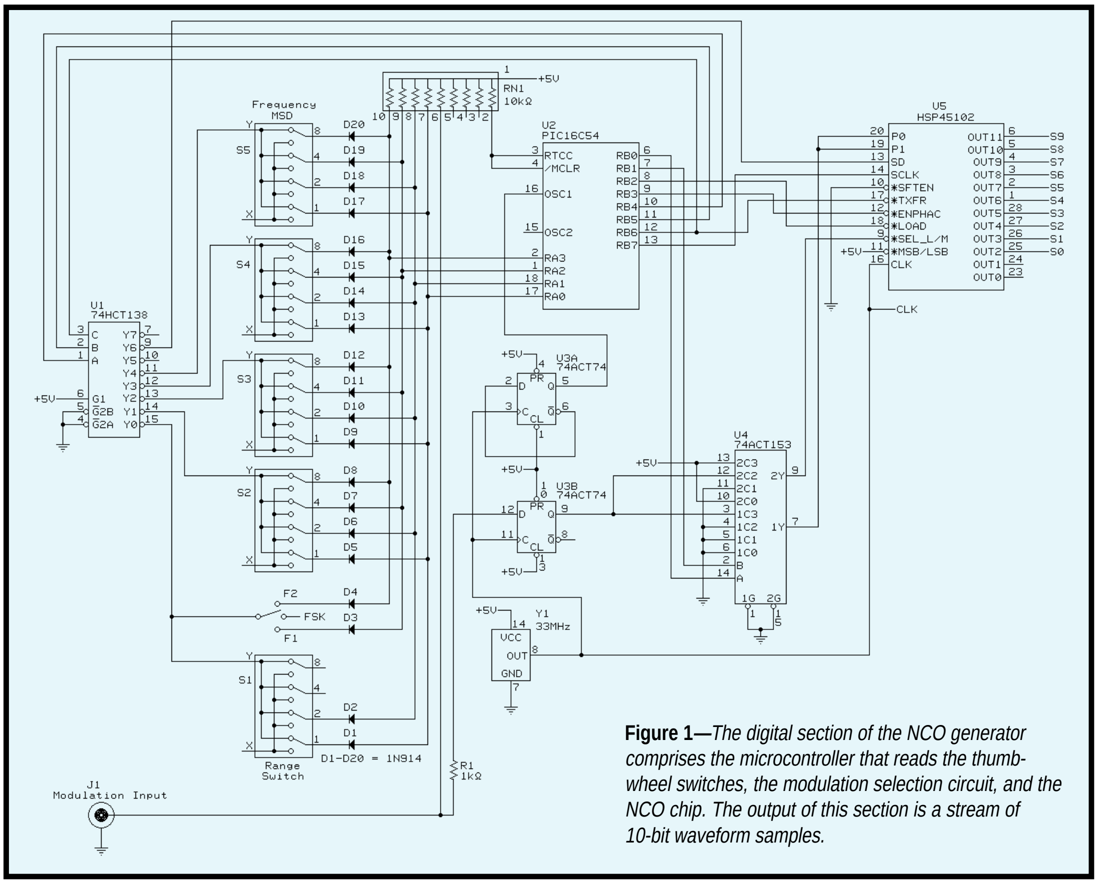
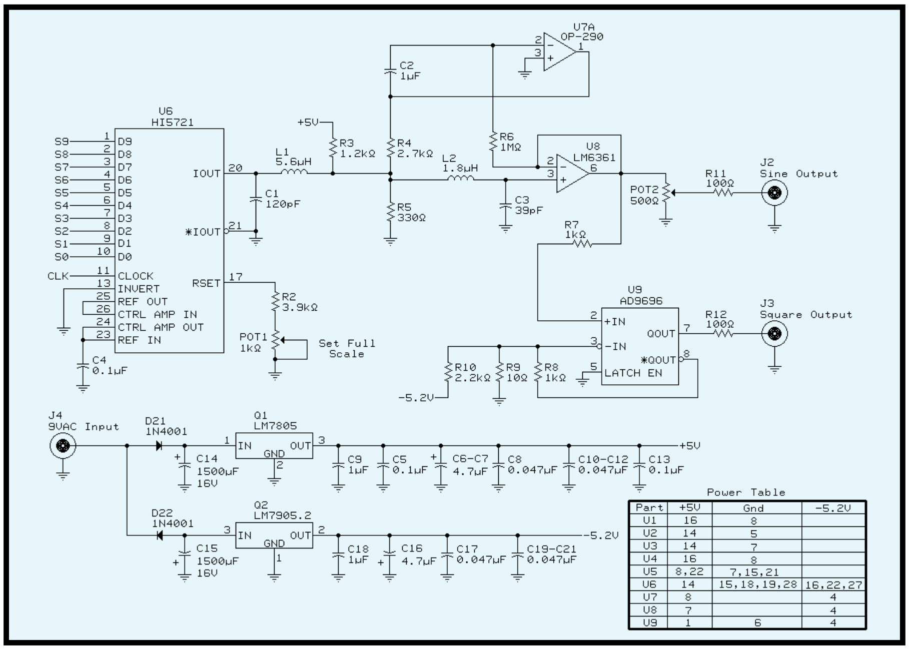


Figure 1—*The digital section of the NCO generator comprises the microcontroller that reads the thumb-wheel switches, the modulation selection circuit, and the NCO chip. The output of this section is a stream of 10-bit waveform samples.*



Part	+5V	Gnd	–5.2V
U1	16	8	
U2	14	5	
U3	14	7	
U4	16	8	
U5	8,22	7,15,21	
U6	14	15,18,19,28	16,22,27
U7	8		4
U8	7		4
U9	1	6	4

Figure 2—*The analog section of the NCO generator includes the DAC, a low-pass filter, the DC-restoration circuit, and the optional comparator chip. It also includes the DC power regulators.*

The 74AC series chips also work, but the 74HCT series is a little slow.

CONSTRUCTION

I built the generator on a Radio Shack universal board and housed it in a metal utility cabinet (see Photo 1). Power comes from a 9-VAC wall transformer that drives two half-wave rectifiers, reservoir capacitors, and +5- and –5.2-V regulators. These are bolted to the back panel.

An adjustable negative regulator can be substituted if a fixed –5.2-V regulator is unavailable. A –5.0-V part is a bit close to the DAC manufacturer's specified minimum supply voltage.

I used a surplus thumb-wheel switch bank that came with a decimal-point position and power-of-ten indicator. Distributors such as Digi-Key sell thumb-wheel switches in various sizes.

Since this circuit is built on a board without a ground plane, good grounding, power-supply pin bypass capacitors, and short direct connections are a must. I used DIL sockets with built-in bypass capacitors for the digital components and mounted bypass capacitors inside the open-frame 28-pin NCO and DAC sockets.

I used some surface-mount capacitors to bypass the power pins of the analog parts. Regular 0.1-µF capacitors will do if their leads are kept very short.

If you can find tinned copper bus strip, use it. It also acts as a shield between the components. Otherwise, use thick bus wire for grounds.

I had some 0.156″ spacing prototyping board that matched the pin spacing on the thumb-wheel switches, so I used it as a bus connection between the diodes I soldered to the switch pins. I then connected ribbon cable from this bus to a 16-pin DIL connector.

This cable mates with a second 16-pin connector on the board and lets the circuit board be removed from the box. I wired the power supplies via the same connector.

The power-supply parts are bolted or glued to the back panel. The –5.2-V regulator needs to be insulated from the panel by a mica washer.

Twisted-pair wire connects the board to the output connectors. Since there's no guarantee the generator will be used in a 50-Ω system, the wiring impedance isn't critical. Series resistors in each output protect the circuit from short circuits and give source impedance matching to reduce reflections if a high-impedance load is connected.

USING THE OUTPUT

There are two ways to terminate the outputs. For one, you can drive into a high impedance, in which case you get the maximum available output voltage but the cable capacitance attenuates the higher frequencies.

Alternatively, you can use 50- or 75-Ω cable with the corresponding

Listing 1—*Column 1 shows the hex address every eight instructions. The remaining columns are the 12-bit instructions.*

```
00    C00  006  C07  026  205  E0C  1E2  C00
08    026  000  A1C  C07  026  C10  A16  C05
10    026  C14  A16  C06  026  A04  03E  9B0
18    92E  93F  966  A04  070  071  072  074
20    075  076  C40  033  CC0  037  966  205
28    E0C  743  A04  C01  1A6  A27  C05  02F
30    C08  024  C0F  166  384  E70  126  C0A
38    9B8  205  020  2A4  2EF  A32  A7B  21E
40    024  C04  02F  060  2A4  2EF  A43  208
48    02D  36D  36D  C0C  16D  C09  03D  C04
50    02F  9BC  038  9BC  039  9BC  03A  9BC
58    03B  21D  024  280  03C  0FC  643  A62
60    97E  A5D  2BD  2EF  A51  800  C08  02F
68    C17  024  C60  126  200  02D  C08  02E
70    486  6ED  586  5E6  4E6  36D  2EE  A70
78    0E4  2EF  A6C  C0F  166  800  21E  F10
80    743  A99  218  1F0  703  A8B  3F1  A8B
88    3F2  A8B  2B3  219  1F1  703  A92  3F2
90    A92  2B3  21A  1F2  603  2B3  21B  1F3
98    800  218  1F4  703  AA2  3F5  AA2  3F6
A0    AA2  2B7  219  1F5  703  AA9  3F6  AA9
A8    2B7  21A  1F6  603  2B7  21B  1F7  800
B0    C6A  02F  06E  2EE  AB3  2EF  AB3  800
B8    02E  2EE  AB9  800  20D  2AD  1E2  883
C0    800  800  800  81F  805  800  800  833
C8    833  800  800  800  800  802  800  800
D0    800  814  800  800  800  8C8  800  800
D8    800  8D0  807
```

matching resistor. This option reduces the output amplitude to about a third but lets even the highest output frequencies reach the device being driven.

FIRMWARE

The firmware is simple since the PIC micro has only two main tasks. It reads the thumb-wheel switches, converts that information into a frequency control number by indexing into a decade parameter look-up table, and sends the result to the NCO.

It also checks the position of the mode switch and sets up the multiplexer. This switches the modulation input to the appropriate NCO pins. In FSK mode, only the mode switch is read. No frequencies are loaded to the NCO.

The firmware comprises 220 instructions. The raw code is laid out in Listing 1 and can be typed into any PROM programmer that can handle PIC microcontrollers.

FREQUENCY COMPUTATION

The frequency-control algorithm is simple as well. Each digit of the thumb-wheel switch is assigned a decimal weight that depends on the position of the four-way range switch.

There are seven possible weights, ranging from one to one million. Each weight has a corresponding 32-bit frequency-control value, which is the weight times either 128 or 131.072 (33.554- or 32.768-MHz crystal, respectively). The table of seven 32-bit weights is precomputed for the appropriate crystal.

The frequency-control number is computed by taking four successive 32-bit table entries (the start point depends on the range) and adding each to a 32-bit total *n* times, where *n* is the corresponding switch digit. This task requires many 32-bit additions but avoids 32-bit × 4-bit multiplications.

Photo 1—*The majority of the generator is built on a 3″ × 4″ prototyping board. The empty socket makes the connection to the thumb-wheel switches.*

GETTING IT RUNNING

Testing this generator requires at least a voltmeter and oscilloscope with a bandwidth in the 50-MHz region. Start with a chipless board so you can test as you go, minimizing the risk of expensive damage.

First, hook up the power and test that all power pins show the correct voltages (either +5 or –5.2 V). Now, you can insert the crystal and the 74ACT74 chip. (Always switch off the power before inserting chips!)

Test for the crystal output on pin 16 of the NCO socket and pin 11 of the DAC socket. Pin 16 of the PIC socket should show half the crystal frequency.

Put the mode switch in one of its extreme positions, and plug in the programmed PIC chip. All four Port A bits (pins 1, 2, 17, and 18) should be high.

Now, you can plug in the 74HCT-138 decoder. Pins 14, 13, 12, and 11 should show sequential negative pulses (each ~10 µs long), which are the switch selector pulses. The PIC's Port A bits should now show a data pattern that changes with the thumb-wheel switch setting.

Check the most significant bit of the PIC's Port B—pin 13. It should show a burst of 64 positive pulses occurring in groups of eight. Each pulse is ~200 ns wide. The burst is ~150 µs long and repeats every 20 ms.

Verify that the burst is also present on pin 14 of the NCO socket. A burst of data should be visible on pin 13 of the NCO socket. Between bursts, pin 13 will be high.

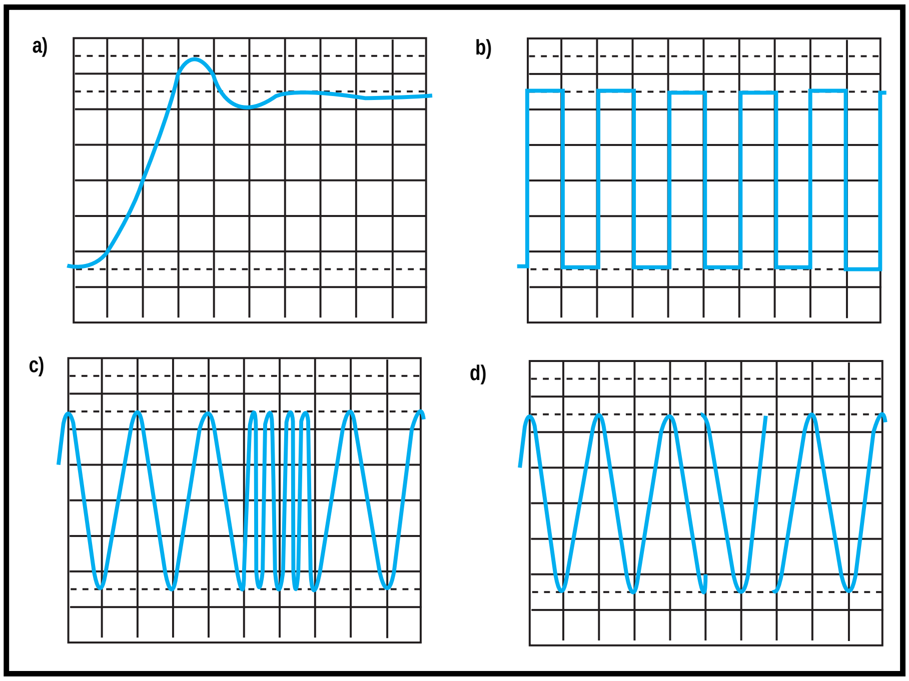


Figure 3a—*The output rising edge is in test mode. The horizontal scale is 20 ns per division.* ***b—****The generator's square wave output is clean at 5 MHz.* ***c—****In frequency shift keying (FSK), the binary modulation switches the sine wave between two frequencies without any amplitude discontinuities.* ***d—****Biphase modulation (BPSK) inverts the sine-wave output whenever the binary modulation input is a 1. This can generate large steps in the waveform.*

Hook a wire between the outer two pins of the mode switch, which should still be in an extreme position. Check that pin 18 on the NCO socket is low and that pin 9 shows a 250-kHz TTL-level square wave. Pin 9 should go high when the mode switch is put in its central position.

Power down, insert the NCO chip, and put the mode switch back in an extreme position. Check that pins 1–6 and 24–28 show a 250-kHz square wave. Don't worry if the signal's high states are curved.

Next, remove the jumper on the mode switch, put the switch in its low position, and set the thumb-wheel switches to 1000E0. The NCO's pin 6 should show a 1-kHz square wave.

If all is well, changing the thumb-wheel setting should generate the appropriate square-wave frequency.

Test each switch digit in turn by zeroing all other switches and checking that the output frequency steps each time the switch under test is moved. This task confirms that the switch connections and decoding logic are in order and that none of the switch diodes are installed backwards.

On the highest range, the output on pin 6 shows a lot of jitter, but this situation is normal. Indeed, it's why an NCO isn't used to generate a square wave directly.

Now, plug in the DAC. Reset the thumb-wheel switches to 1000E0, and leave the mode switch low. If everything is hooked up properly, DAC pin 25 should be near –1.25 Vpp. A scope probe on DAC pin 20 should show a 1-kHz sine wave with about a 1-Vp-p amplitude, centered roughly about 0 V.

If everything works, power down and plug in the LM6361 and the OP-290. On powering up, the same 1-Vp-p signal should be visible on pin 6 of the LM-6361 and at the BNC sine-output socket (if the amplitude control is set to full).

The output should now be equally balanced about ground or at least heading in that direction. It can take tens of seconds to reach an exact balance after turning on.

Put the unit back in test mode with the jumper across the mode switch. The output signal on the amplifier (pin 6) should now be a ±0.5-V square wave at 250 kHz. Adjust the amplitude trimmer until the output is correct.

Check that the square wave's leading edge looks like Figure 3, which has a horizontal scale of 20 ns per division. The edge should have a rise time of 30 ns with no more than 12% overshoot.

The waveform in Figure 3a represents the BNC output connected via 50-Ω cable to a 50-Ω input oscilloscope.

With the test jumper removed, it should be possible to set any output frequency and see a clean 1-Vp-p sinewave output. The output amplitude will roll off a little above 5 MHz.

Above 1 MHz, there will be a little fuzziness in the waveform caused by the residual alias frequencies passing through the filter. You can see how bad this effect is by setting a frequency that is a power of two times 1 kHz (e.g., 4096 kHz).

At these frequencies, the aliases are in phase with the wanted signal and show up as a small static ripple in the waveform. They should be barely visible.

Two things remain to be tested—modulation and the square-wave output. For the latter, plug in the AD9696 comparator, and check that the square-output BNC connector shows a TTL-level square wave at the frequency set by the thumb-wheel switches. At 5 MHz, it should look like Figure 3b.

Modulation testing requires an external TTL pulse generator, ideally one with an accurate crystal timebase. With the external generator connected to the modulation-input BNC connector and the mode switch high, the output sine and square waves will be 180° phase modulated by the TTL input.

The effect is most easily seen if the scope is triggered from the pulse generator and the NCO generator's frequency is set to a small multiple of the modulation frequency. The sine wave's drift rate is a function of the frequency error between the NCO and pulse-generator clocks (see Figure 3c).

If the mode switch is set low and the frequency changed, setting the mode switch to its center position causes the output to switch between the two frequencies according to the modulation input. Figure 3d is a typical example.

Since the modulation input has a pull-up resistor, it can be switched high and low by a simple on/off switch connected to the socket. This switch can be handy when a test requires either of two fixed frequencies.

AREAS TO EXPLORE

The beauty of an NCO generator is the variety of things you can do with it. It generates frequencies from below the audio range up to the short-wave bands.

If you use a bigger box, you can add digits to the frequency-setting switch and achieve better frequency resolution.

If you're ambitious, you can run two NCOs at the same frequency but with different phases (e.g., to perform quadrature modulation of a carrier).

NCOs running at different frequencies can have their outputs added or multiplied together before being sent to the DAC. This way, you can demonstrate many interesting mixing and modulating schemes.

Many thanks to the applications engineers at Harris Semiconductor for their help and to Newark Electronics for providing samples.

Tom Napier has worked as a rocket scientist, health physicist, and engineering manager. He spent the last nine years developing space-craft communications equipment but is now a

consultant and writer. You may reach Tom via E-mail at eric@voicenet.com.

SOFTWARE

The source code, in official Microchip mnemonics, is available from the Circuit Cellar Web site.

SOURCES

HSP45102, HSP45106, HSP45116, HI-5721BIP, HI-5731 DAC
Allied Electronics
7410 Pebble Dr.
Fort Worth, TX 76118
(817) 595-3500
Fax: (817) 595-6406
www.allied.avnet.com

Harris Corp.
1025 W. Nasa Blvd.
Melbourne, FL 32919
(407) 727-4000
Fax: (407) 724-3973
www.harris.com

Newark Electronics
12880 Hill Crest Rd.
Dallas, TX 75230
(972) 458-2528
Fax: (972) 458-2530
www.newark.com

DACs and fast amplifiers
Analog Devices, Inc.
One Technology Way
Norwood, MA 02062-9106
(617) 329-4700
Fax: (617) 329-1241

Thumb-wheel switches, connectors, inductors, amplifiers, 74ACT chips, PIC microcontrollers
Digi-Key Corp.
701 Brooks Ave. S
Thief Falls, MN 56701-0677
(218) 681-6674
Fax: (218) 681-3380

PLP-10.7
Mini-Circuits
13 Neptune Ave.
Brooklyn, NY 11235-0003
(718) 934-4500
Fax: (718) 332-4661
www.minicircuits.com

LM6361
National Semiconductor
P.O. Box 58090
Santa Clara, CA 95052-8090
(408) 721-5000
Fax: (408) 739-9803

General-purpose NCOs
Stanford Telecom, Inc.
480 Java Dr.
Sunnyvale, CA 94089
(408) 745-2660
Fax: (408) 341-9030
www.stelhq.com

PIC16C54
Microchip Technology, Inc.
2355 W. Chandler Blvd.
Chandler, AZ 85224-6199
(602) 786-7200
Fax: (602) 786-7277
www.microchip.com

Preprogrammed PIC16C54 .... $8
Tom Napier
P.O. Box 3155
Maple Glen, PA 19002-8155

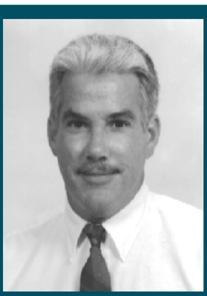
I R S

425 Very Useful
426 Moderately Useful
427 Not Useful

SILICON UPDATE

Tom Cantrell

M•Core on the March



To Tom, M•Core offers PowerPC performance and the cost, power, and code density of the 68k. It's a 32-bit RISC that delivers now and leaves plenty of headroom for the future.

Treading down the primrose 32-bit processor path isn't something to be done lightly. Step gingerly past the remains of chips like the 88k, 29k, 860, and others that fell along the wayside.

Certainly, the desktop CPU market is effectively closed at this point with '*x*86 dominant, PowerPC (and Apple) a dark horse, and the familiar collection of workstation RISCs (e.g., MIPS, SPARC, Alpha, and HP-PA) watching NT warily. New desktop CPU architectures need not apply.

The embedded world is another story. The breadth of the application demands more not fewer choices—everything from 4 to 64 bits. Yes, this includes embedded PCs and stripped versions of the desktop RISCs, but there's still plenty of room for embedded-only 32-bit chips like the ARM and Hitachi SH.

Leave it to Motorola to keep the proud tradition of processor proliferation alive with M•Core, a 32-bit streamlined RISC. According to Motorola, M•Core is more than a brand-new chip. Coincident with their latest reorg, it represents a whole new way of doing business.

In essence, M•Core consists of intellectual property including the core design, third-party and in-house tools, documentation, EV boards, and so on. This know-how can be turned into chips in a number of ways as illustrated in Figure 1.

For instance, one of the first M•Core customers is combining the CPU with a DSP core to handle base-band processing in a cellular phone.

Motorola's semiconductor group can add M•Core to their stable, too, using it as the nucleus of standard products and adding it to their ASIC cell library. It's conceivable that M•Core might be even more widely proliferated, perhaps to other IC suppliers.

REDUCED RISC

Although M•Core adopts certain aspects and philosophies of other Motorola architectures (e.g., the 68k, ColdFire, and PowerPC), they're combined in such a way that the end result is quite distinguishable.

M•Core is perhaps best described as aspiring to PowerPC-like performance, while offering the cost, power, and code-density advantages of chips like the 68k and ColdFire. As you see in Figure 2, it's a moderately aggressive 32-bit RISC that delivers now and offers plenty of headroom for the future.

Initially, M•Core relies on a four-stage pipeline, arguably right in the middle of the three- to five-stage sweet spot that delivers the best bang per buck. That is, it offers high performance at moderate (i.e., double digit) clock rates without a lot of circuit complexity. This class of design is well-understood, as are possible future enhancements like superscalar techniques, high-speed numerics, and faster clock rates.

Unlike chips that take excessive liberty with the RISC moniker, M•Core is relatively true to form. For instance, it's a pure load/store architecture. Instructions only operate on registers, except for load and store, which are the only instructions that access memory.

The programmer's model is kind of a hybrid of PowerPC and 68k with a bit of Z80 thrown in for good measure (see Figure 3). The chip features sixteen 32-bit registers, which are mostly general-purpose (R0 and R15 serve double duty as data and return address stack pointers). A second bank of registers enables high-speed interrupt response, while the 68k-like User/Supervisor

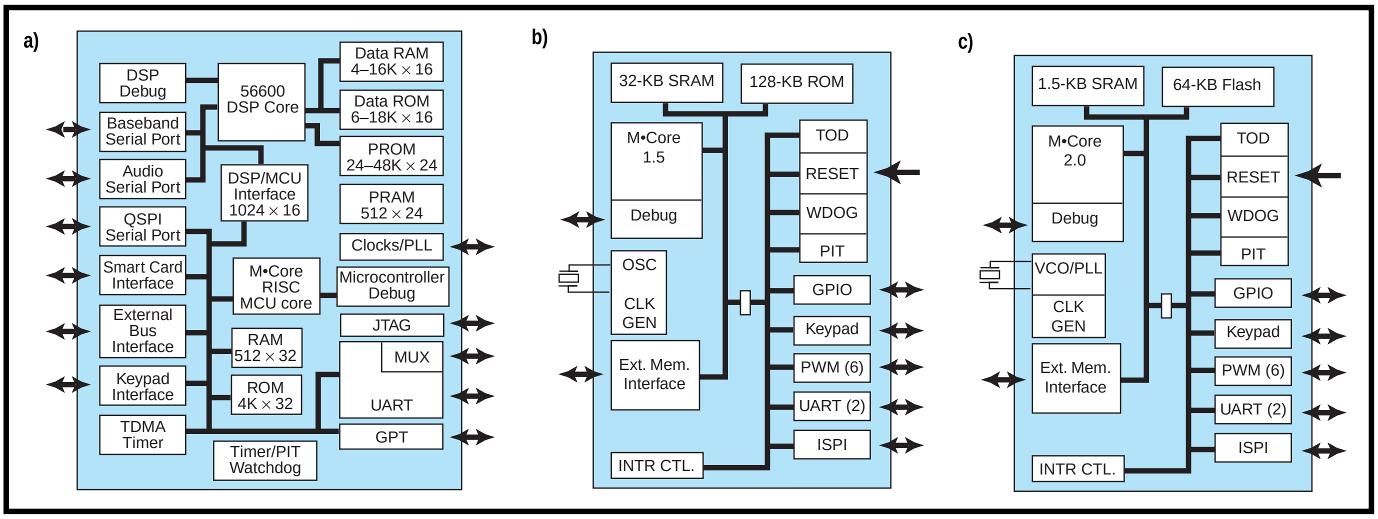


Figure 1a—*Motorola offers M•Core for ASICs such as a cellular phone chip.* ***b & c***—*Motorola also plans standard products for commercial and industrial applications.*

protection scheme exposes 13 other specialized registers to trusted software.

TIGHT CODE

RISCs usually feature fixed-length instructions (faster and easier to pipeline and superscale), and M•Core follows suit. Traditionally, most chips devote 32 bits to the cause—especially those with large register files and/or three operand instructions, both of which consume opcode bits at a prodigious rate.

In the performance-at-any-price desktop world, poor code density isn't a big deal. First, in a system with CRTs, disks, printers, and other expensive iron (not to mention pricey CPUs), memory isn't the main cost factor.

Second, even a 50–100% code-size penalty washes out in light of the overall code-bloat trend. Consider the example of the RISC Mac versus the CISC PC. Code density isn't high on the list of factors considered when choosing between the two.

But in the embedded world, especially the portable battery-powered segment, code density deserves more thought. Memory cost may represent a much higher, if not the highest, portion of system cost.

Poor code density may compel the addition of memory chips and complicate single-chip integration. Equally critical—and totally unlike desktop machines—memory power consumption can't be ignored.

M•Core finesses the performance/code-density tradeoff by going with fixed-length instructions, but ones only 16 bits wide. It's not surprising that other machines with tight encodings such as the Hitachi SH, ARM Thumb, and MIPS16 (the latter two using 16-bit opcode subset schemes) represent the most likely M•Core competition.

Some RISCs use delay slots so nondependent instructions execute during the branch and load delays. However, delay slots worsen code density since the compiler must insert NOPs when it can't find a useful instruction to schedule—an all-too-often occurrence.

A more subtle problem is that assembly-language programming and debugging (still necessary in many real-time apps) is complicated when instructions get moved around.

By contrast, M•Core avoids delay slots entirely. Yes, loads and branches incur a pipeline stall, but the payback is elimination of superfluous NOPs, WYSIWYG assembly language, and simpler faster compilers.

INSTRUCTIONS APLENTY

Though the instructions may be short, there's no shortage of them with nearly 100 different opcodes by my count. However, that's a bit misleading since many are just minor variations on a theme.

For instance, you may have noticed in Figure 3 that the USER status info consists of a single C (carry) bit. Instead of having a lot of different status bits, M•Core offers multiple types of compare instructions (e.g., higher, equal, not equal, less than, etc.) using the C bit as a universal condition code.

Besides expected use directing conditional branches, the C bit also supports a form of conditional execution

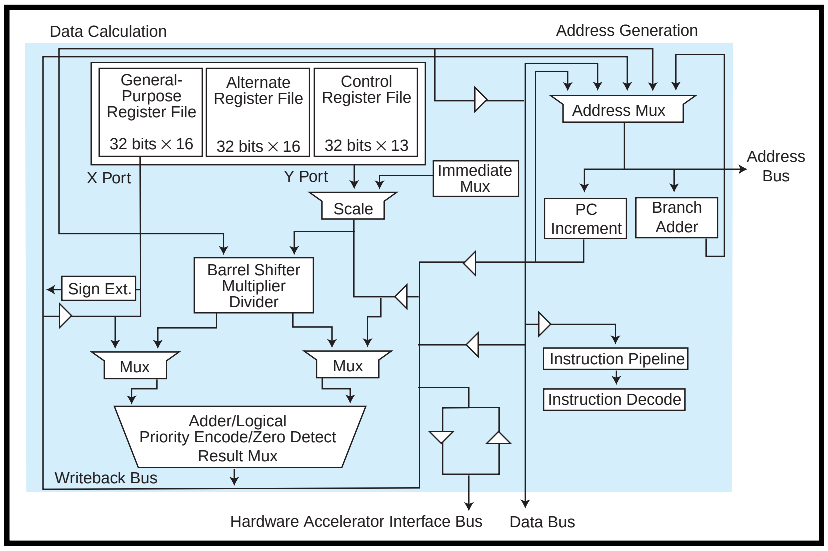

Data Calculation
Address Generation
General-Purpose Register File 32 bits × 16
Alternate Register File 32 bits × 16
Control Register File 32 bits × 13
X Port
Y Port
Immediate Mux
Scale
Address Mux
Address Bus
PC Increment
Branch Adder
Sign Ext.
Barrel Shifter Multiplier Divider
Mux
Mux
Adder/Logical Priority Encode/Zero Detect Result Mux
Writeback Bus
Instruction Pipeline
Instruction Decode
Hardware Accelerator Interface Bus
Data Bus

Figure 2—*M•Core tempers conventional RISC wisdom with 16-bit opcodes, powerful bit processing, and programmer-friendly (i.e., invisible) pipeline.*

for selected operations including register increment, decrement, clear, and move. Conditional execution shrinks code and also equalizes execution time regardless of conditional evaluation.

Per RISC dogma, most M•Core instructions (including 1–32-bit shifts) execute in a single clock, thanks to a built-in barrel shifter. In fact, M•Core adds a number of other bit-oriented features likely to be useful in I/O-oriented applications.

Besides the expected bit-set and -clear instructions, there two forms of bit-generate instructions. The "immediate" version (BGENI) sets any bit in a register (like bit set) but also clears the rest of them.

The "dynamic" form (BGENR) works similarly, except the bit position to be set is specified in another register rather than immediately. There's also a bit-mask-generate instruction (BMASKI) that works like BGENI, except it sets all bits up to the specified position and clears the rest.

A novel bit-reverse instruction does just what it says (i.e., bit 0 of a register moves into position 31, bit 1 into 30, etc.). Yeah, you may not need to do it often, but BREV is a heck of a lot easier (and faster) than the conventional 32-iteration shift, check, and stuff loop.

Also unique is a Find First 1 (FF1) instruction that returns the position of the first 1 found in a register scanning from most to least significant bits. BREV offers a handy solution if you prefer an least significant bit-first scan.

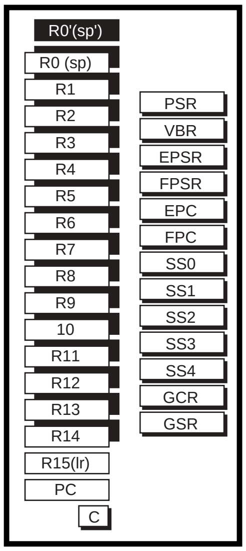


Figure 3—*Dual general-purpose register banks bypass memory bottlenecks and boost real-time performance. A third set of control registers is accessible in supervisor mode.*

Although there's no provision for alignment, bus sizing, or Big versus Little Endian in the core itself, in principle, such support circuits could be added externally. Once onboard, however, the same friendliness M•Core exhibits towards bits extends to bytes and half words, with instructions to load and store as well as extend them (zero and signed) to 32 bits.

There are also four byte-specific extract instructions (i.e., XTRB0–3) that copy the specified byte from one register to the low-order byte (with zero extend) of another. Finally, there's an instruction (TSTNBZ) that checks whether any byte in a register is 0.

Like most RISCs, not every M•Core instruction executes in a single clock. As I mentioned, loads, stores, and taken branches require two clocks.

It's also predictable that MULT and DIV take longer. The former delivers two bits per clock with two clocks overhead (e.g., a 16 × 16 multiply takes 10 clocks, 32 × 32 takes 18 clocks). DIV (unsigned and signed versions) takes 3–37 clocks depending on the result magnitude using an early-out algorithm.

There are other multi-clock instructions where the extra clocks are an unavoid-

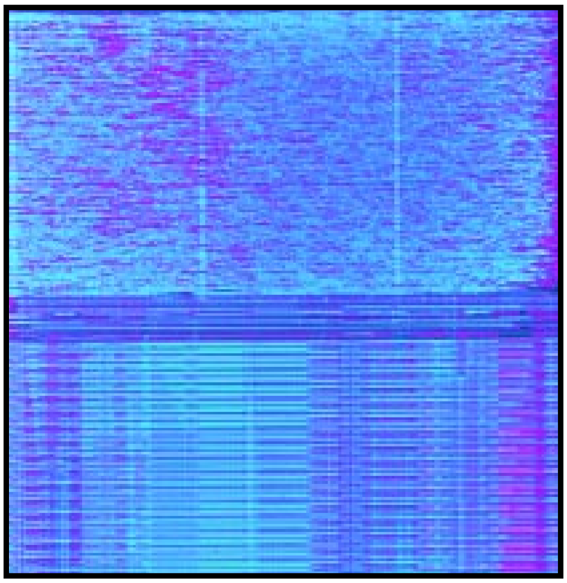


Photo 1—*Using the latest design (synthesis) and production (0.36 µm) know-how, M•Core cuts cost and power use. The core itself (i.e., no memory or I/O) is only 2.2 mm^2 and consumes less than 1.5 mW/MHz at 3.3 V, targeting 0.5 mW/MHz and 1.8 V in the future.*

Table 1—*M•Core exception handling recalls the 68k with some key improvements. Note the vectors for both normal (INT) and fast (FINT) interrupts and provision for add-on hardware accelerators.*

Vector Number(s)	Vector Offset (hex)	Assignment
0	000	Reset
1	004	Misaligned Access
2	008	Access Error
3	00C	Divide by Zero
4	010	Illegal Instruction
5	014	Privilege Violation
6	018	Trace Exception
7	01C	Breakpoint Exception
8	020	Unrecoverable Error
9	024	Soft Reset
10	028	INT Autovector
11	02C	FINT Autovector
12	030	Hardware Accelerator
13	034	(Reserved)
14	038	(Reserved)
15	03C	(Reserved)
16–19	040–04C	TRAP #0–3 Instr Vect
20–31	050–07C	(Reserved)
32–127	080–1FC	Reserved for Vect Int Controller Use

able by-product of data movement. For instance, there are multiregister versions of load and store.

LDQ/STQ moves the quadrant of registers comprising R4–R7, while LDM/STM moves a contiguous range from R*n*–R15 to and from the stack. Note that *n* must be between 1 and 14, since R0 is the stack pointer and a LDM/STM R15-R15 is better handled as a simple LD/ST R15.

These instructions only take one plus the number of register clocks (e.g., moving five registers = six clocks), making them faster (and shorter) than a sequence of individual two-clock LD/ST instructions. Data access time also affects a few other instructions, such as indirect branches and returns (three clocks) and TRAP (five clocks).

Power-management embellishments include three low-power instructions—STOP, WAIT, and DOZE. However, since M•Core defines a core, not a complete chip, it simply signals which if any of these instructions have been executed, leaving the exact function (e.g., stop the clock) to external circuits.

EXCEPTIONAL EXCEPTIONS

For all their bandwidth, traditional RISCs sometimes don't handle distractions (in the form of interrupts and exceptions) well. M•Core builds on the familiar and proven 68k exception model, adding a number of enhancements to improve real-time response.

Listing 1—*Dedicated registers (R12–14) and the FF1 (Find First 1) instruction form the basis of a software-driven interrupt priority scheme. FF1 identifies the highest priority pending interrupt. The most significant bit of a priority word in memory corresponds with the highest priority (e.g., UART_TX), and the least significant bit with the lowest priority (e.g., SW3).*

```
; Assumes the following alternate register file image:
;   r14—pointer to priority word
;   r13—pointer to Normal Vector Table
;   r12—scratch register
Normal_Service:
  ld      r12,(r14)          ; get contents of priority word
  ffl     r12                ; find highest priority intr
  lsli    r12,2              ; translate to offset
  add     r12,r13            ; point to assoc. table entry
  ld      r12,(r12)
  jmp     r12                ; jump to intr service routine
  .align  WORD_ALIGN
; Indexed vector table contains addresses of each of the
; 32 service routines. FFI instruction returns offset from
; most significant bit of the register
Normal_Vector_Table:         ; addresses of service routines
  .long   UART_Tx_Service
  .long   UART_Rx_Service
  .long   Timer_Service
  …
  .long   SW3_Service
  .long   Spurious_interrupt_Service
XXX_Service
  …
  rte
```

Clearly, the extra registers are at the top of the list, since they can eliminate or reduce the time-consuming chore of saving and restoring context.

The function of the alternate register file (i.e., R0′–R15′) is obvious. Its selection is controlled by a bit in the processor status register (PSR). Though the control bit is only accessible in supervisor mode, both supervisor- and user-mode programs can access either file.

Let's take a closer look at the extra supervisor control registers (refer back to Figure 3) since they play a crucial role in exception processing.

The lineup starts with the PSR, which contains the myriad bits that control exception-related operations. Exceptions (listed in Table 1) may occur as a result of internal operation or external interrupt request.

M•Core defines two interrupt exceptions—normal and fast, the latter having higher priority. The PSR contains three bits that enable or disable exceptions, normal interrupts, and fast interrupts. Another bit controls whether multiclock (e.g., MULT, DIV, LDM/STM, etc.) instructions can complete or terminate early (and subsequently restart) when an interrupt is pending.

Like the 68k, M•Core relies on a vector table in memory, and VBR allows the table's base address to be changed dynamically. Also like the 68k, external interrupt requestors (both normal and fast) can either provide a vector or accept the default autovector.

The least significant bit of a vector table entry isn't needed since instructions must be 16-bit aligned (i.e., it's presumed 0). Instead, that bit automatically selects which register file should be used by the handler.

EPSR and EPC are one-level stacks where the status register (PSR) and PC are saved when an exception occurs. It's up to you to put them somewhere else (typically on the stack) if necessary, lest the next exception overwrite them and force your system to punt.

FPSR and FPC perform the same role but specifically for the fast interrupt request. Thus, you can safely take a fast interrupt during exception processing without muss and fuss.

Five extra storage registers (SS0–4) are handy for holding key pointers and parameters. So, each can store the stack address for a different task, swapped in and out of R0 for each task switch.

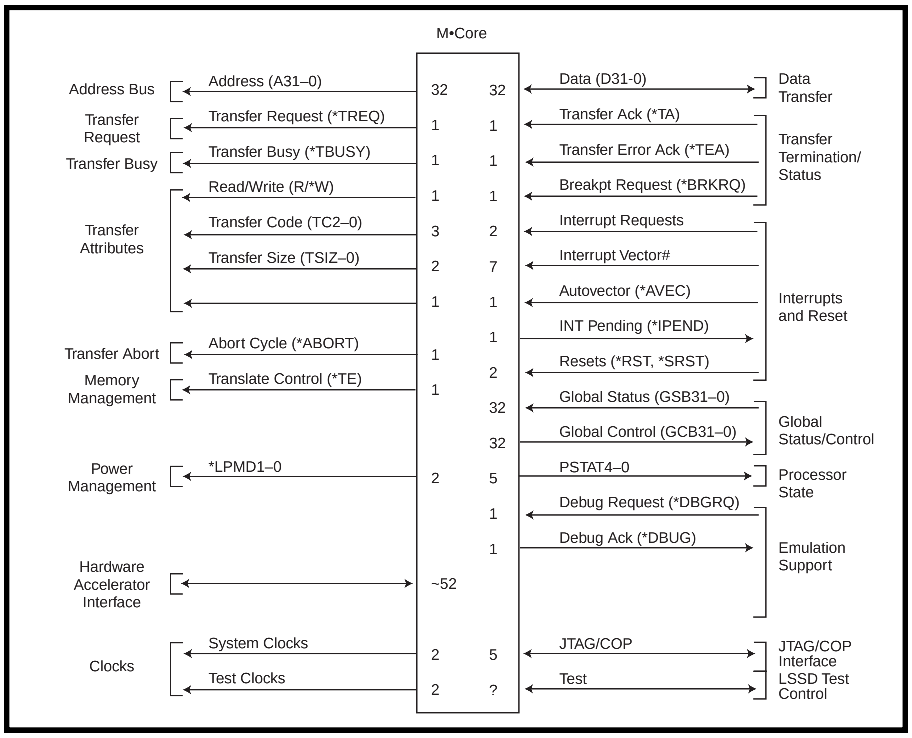


Figure 4—*Though intended for on-chip connection, the M•Core bus is similar to a conventional CPU, mainly differentiated by the addition of a dedicated hardware accelerator interface.*

Since M•Core doesn't define I/O and glue logic around the processor, the GSR and GCR provide a direct path (32-bit input and output, respectively) to the CPU for these add-ons.

Of course, with no a priori knowledge of a particular implementation, M•Core can't offer the comprehensive interrupt priority schemes of completely integrated chips. However, software can do a pretty good job as Listing 1 shows.

This routine exploits the alternate registers and FF1 instruction to quickly prioritize interrupts from a hypothetical selection of hardware. After a minimum exception latency of six clocks (i.e., EPSR, EPC save, vector table fetch, branch), it only takes nine clocks to find the highest priority interrupt and execute the handler's first instruction.

HW XLR8R

Figure 4 shows the "pinout" of M•Core, keeping in mind that circuits directly connected to the core are usually integrated on the same chip. Although Motorola plans to offer a raw M•Core CPU, the 384-pin chip shown in Photo 1 is really intended for evaluation and prototyping.

The primary interface signals (i.e., address, data, control, interrupts) are self-explanatory, especially if you know the 68k. As with that chip, an asynchronous transfer protocol is used so the system must return active acknowledgment (i.e., *TA or *TEA) to complete a transfer.

However, the intimacy of connection at the core level calls for extra interface logic. For example, the core simply outputs a *SEQ (sequential) indicator. You handle the details of a particular memory-burst transfer scheme.

The same goes for memory management. The CPU outputs a translate enable (*TE) signal for an external MMU.

As mentioned, the execution of low-power mode instructions (LPMD0-1) is left for the external logic. The core offers a bit of support for leaving low-power mode with the *IPEND (interrupt pending) output which, derived from the level-sensitive interrupt inputs, works even when the clock is stopped.

Access at the core level is further distinguished by a unique hardware accelerator interface. At the time I'm writing this, some details are TBD, but what is known is worth examining.

Consisting of a separate 32-bit data bus and miscellaneous address and control lines, the accelerator interface is set up for tight coupling to on-chip intelligence like a DSP or other coprocessor.

For a simple output-only connection, just register snoop. With this, changes to M•Core registers (primary or alternate) are echoed externally (i.e., 32-bit data, 5-bit register specifier, *REGWR strobe). External hardware latches output, making a shadow copy of the CPU register.

You can also spread smarts by passing instructions. M•Core reserves opcodes for this and outputs them and a signal from the instruction decoder when they're fetched. Once the coprocessor signals readiness, M•Core issues the order to execute.

Passing data with instructions likely involves transfer primitives, such as accelerator call/return and load/store. The former count the registers to be transferred (up to seven, starting with R4), while the latter bridge data across the system and accelerator interfaces.

Core-level interface also inspires an M•Core debug built on various hooks, including a software- and hardware-invocable debug mode, a breakpoint request input (for instructions and data), JTAG or other serial scan scheme (access to registers, etc.), and overt clock control. Host software uses some or all of these functions as the foundation for a sophisticated development suite.

NICE CHIPS FINISH LAST

It's the nature of progress—and hindsight—that the latest chip is often the best. M•Core is no exception. It delivers good performance efficiently.

Ultimate success depends on more than features, though. Motorola has to deftly position M•Core against worthy competitors (including their own 68k, ColdFire, and PowerPC).

One challenge is third-party tools. Achieving the quality and breadth of support of, for example, a 68k doesn't happen overnight.

But, M•Core's off to a good start. Its quiver includes Diab Data C/C++, SDS simulator/debugger, HP logic analyzer probes, and ISI and Microtec RTOSs.

Given this initial lineup, it's reasonable to believe M•Core will achieve the critical mass needed to be a contender.

Without a steady supply of new chips, the IC revolution would be cut short. I might end up selling insurance or something equally grim.

So, thanks Motorola. Keep the chips coming, and keep hope alive. ◘

Tom Cantrell has been working on chip, board, and systems design and marketing in Silicon Valley for more than ten years. You may reach him by E-mail at tom.cantrell@circuitcellar.com, by telephone at (510) 657-0264, or by fax at (510) 657-5441.

SOURCE

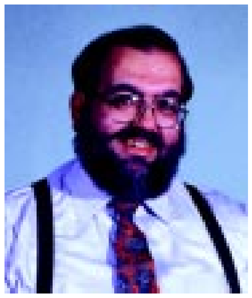
M•Core
Motorola
7600 Capital of Texas Hwy.
Austin, TX 78731
(800) 521-6274
Fax: (800) 521-6274
www.motorola.com/mcore

I R S

428 Very Useful
429 Moderately Useful
430 Not Useful

PRIORITY INTERRUPT

After Ten Years, Inside the Box Still Counts



Writing an editorial for the tenth-anniversary issue is a little like sitting down at the dinner table for a special occasion. Your first inclination is to simply say grace and thank everyone responsible for making the day happen. Certainly, this is no different. Any successful magazine is the work of many people and not just the achievement of a single individual. Still, a tenth anniversary is a cause to celebrate the past, analyze the present, and predict the future.

As for the past, probably the greatest cause for celebration is being successful against the odds. How many magazines have you subscribed to during the last 10 years that no longer exist? I'd like to say we're still here because of our technical prowess and foresight. While there's no doubt they're among the principal reasons you read *INK*, in truth, these attributes are the de facto result of simply letting an engineer choose an editorial course most interesting to other engineers. Our motto is and has always been that *we are a magazine written for engineers, by engineers.*

Of course, today, I've come to interpret "engineer" to more appropriately indicate an engineering cast of mind rather than any strict degree designation. It would be far too egotistical of me to diminish the valuable contribution of professionals and programmers who don't strictly call themselves engineers. My personal programming language may still be solder, but this is simply my choice and not a campaign platform. The technical cross-pollination involved in today's embedded control designs certainly makes such divisions meaningless.

As for the present, what can I say? "Give me the means and I'll control the world!" The options are more varied today than ever. There are dozens of processors, scores of tools, and even more marketing hype as to the direction designers should take. In a world filled with 4- to 64-bit system solutions, I'm hopeful the real applications we document in the pages of *INK* help separate hype from reality.

We'll continue to bridge the gap between trade magazines and technical enjoyment. Believe me, it's no easy task. As you see more of the large semiconductor companies in our pages, consider the value of the achievement. These companies finally discovered that their engineering departments read *INK*. Of course, the marketing guys didn't know this because they never saw any issues lying around. (Leave *INK* around where it could get taken? No way, they take it home!) Of course, trade magazines are easy to find. They're in unread piles on every desk.

As for the future, it sounds trite, but there is a multitude of possible alternatives. If I mimic today's popular press, I'm supposed to be warning you about the dire meltdown associated with the millennium issue. Apparently, when the clock clicks over to January 1, 2000, there will be massive software glitches causing considerable confusion and financial loss. Banks will start posting mortgage interest from 1900, insurance annuities will cease, and ICBMs will head the wrong way.

Personally, I think all this is a bunch of over-anxiety. Forewarned is forearmed. While it's conceivable the rest of the world might not know that 00 follows 99, I can't see that a traffic-light sequencer or a soda-machine coin counter cares what year it is. Much of what we design will be immune (for what isn't, see Scott Lehrbaum's article, "Year 2000 and Embedded PCs"). Nevertheless, I am prepared for the Department of Motor Vehicles to cancel my registration because they can't calculate a negative age. Forewarned doesn't mean anything to the bureaucracy.

All levity aside, I do have one future prediction. It's easy for us to look at the Internet as simply a neat way to run searches and download datasheets. But, that may be a gross underestimation of its potential application in embedded controls. I believe that the power of all those connected resources combined with the rapid evolution in phone and wireless communication technology will define a new Internet that will offer a completely new control methodology. Cars will have dashboards that download maps, upload vehicle performance, and provide customized entertainment. Televisions, pagers, wrist communicators, and perhaps even microwave ovens will contain chips connecting them to TCP/IP. A browser will connect you to the world.

Certainly, the politics of all this connectivity will play a significant role in its adaptation. As for the technology, you can depend on our tenacity to stay the course and tell you how it's implemented. It may have been ten years, but to this day, I believe that the title on our first issue is still a great truth: **Inside the box still counts**. We won't let you down.

Steve

steve.ciarcia@circuitcellar.com