

EMBEDDED PC
MONTHLY SECTION

CIRCUIT CELLAR

INK[®]

THE COMPUTER APPLICATIONS JOURNAL

#100 NOVEMBER 1998

INDUSTRIAL CONTROL

Control via the Internet

Making Buildings Smarter

Embedded Linux

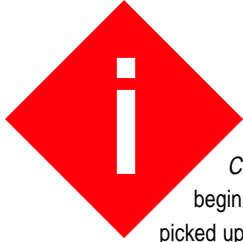
Toughen up
that PC BIOS



\$3.95 U.S.
\$4.95 Canada

TASK MANAGER

Netting 100



In your hands you hold the 100th issue of *Circuit Cellar INK*. Have you been here from the beginning, over 10 years ago? Or, maybe you just picked up this issue from the newsstand today for the very first time. No matter. It's definitely a time for celebration!

I find it fitting that a milestone issue like this one has the theme Industrial Control. After all, home automation and building control has been a Circuit Cellar hallmark from the start. It's nice to see that although the implements have of course changed over the years, the interests haven't. *INK* still focuses on what the readers want.

But, the implements have changed indeed. No longer are the control options limited to switches and sensors. There's a new game in town—well, in the world, to be more accurate: the Internet.

The capabilities we get through the web are far beyond what we've been able to do without it. OK, I admit, sometimes I get a bit bored hearing the same old hype about how everything is going to be done through your computer—you'll turn on the heat at home before you leave the office, you'll order take-out, you'll do some catalog shopping, you'll be able to check whether you left the iron on after you left the house, and on and on. But, I'm starting to get impressed: this scenario is coming true!

In this issue, Chris Sontag shows us a web-implemented irrigation controller system that revises the sprinkler schedule based on weather forecasts retrieved over the Internet. And if the forecasts prove wrong—oh, that never happens, right?—you can simply access the controls via a web browser. (I guess this means no more laughing as you pass the building on the corner that has its sprinklers set for 4 P.M., come rain, come shine.)

Sounds pretty good, right? Of course, there's a new challenge: how are you going to use the Internet in your applications?

Well, if you need some ideas or want to kick around some thoughts with other engineers, here's some good news. *Circuit Cellar INK* is a co-sponsor of the First Annual Embedded Internet Workshop being held on Friday, November 6, at the Wyndham Hotel in San Jose.

This workshop, organized by Dr. Lance Leventhal, offers sessions on embedded web servers, the future of embedded Internet, application development, embedded web hardware, embedded Java, and web security. The day promises to be chock full of information from some of the leaders in embedded web technology—Annasoft Systems, Lucent Technologies, Phar Lap Software, and Phoenix Technologies, to name just a very few.

And of course, we'll be there. Tom Cantrell will introduce keynote speaker Mark Tolliver, who will talk about Java and the embedded market. Steve and Jeff will be attending as well, and I'll be there, actively recruiting editorial. We hope to see you at the workshop, but if you can't make it, we'll make sure you still get to read about the all latest useful control technologies and applications in *Circuit Cellar INK*.

Eli

elizabeth.laurencot@circuitcellar.com

CIRCUIT CELLAR[®] INK[®]

THE COMPUTER APPLICATIONS JOURNAL

EDITORIAL DIRECTOR/PUBLISHER

Steve Ciarcia

ASSOCIATE PUBLISHER

Sue Skolnick

MANAGING EDITOR

Elizabeth Laurencot

CIRCULATION MANAGER

Rose Mansella

TECHNICAL EDITORS

Michael Palumbo
Rob Walker

CHIEF FINANCIAL OFFICER

Jeannette Ciarcia

ART DIRECTOR

KC Zienka

WEST COAST EDITOR

Tom Cantrell

ENGINEERING STAFF

Jeff Bachiochi

CONTRIBUTING EDITORS

Ken Davidson
Fred Eady

PRODUCTION STAFF

Phil Champagne
John Gorsky
James Soussounis

NEW PRODUCTS EDITOR

Harv Weiner

PROJECT EDITOR

Janice Hughes

EDITORIAL ADVISORY BOARD

Ingo Cyliak Norman Jackson David Prutchi

Cover photograph Ron Meadows—Meadows Marketing

PRINTED IN THE UNITED STATES

ADVERTISING

ADVERTISING SALES MANAGER

Bobbi Yush
(860) 872-3064

Fax: (860) 871-0411
E-mail: bobbi.yush@circuitcellar.com

ADVERTISING COORDINATOR

Valerie Luster
(860) 875-2199

Fax: (860) 871-0411
E-mail: val.luster@circuitcellar.com

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](ftp://circuitcellar.com)

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


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.

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

Entire contents copyright © 1998 by Circuit Cellar Incorporated. All rights reserved. Circuit Cellar and *Circuit Cellar INK* are registered trademarks of Circuit Cellar Inc. Reproduction of this publication in whole or in part without written consent from Circuit Cellar Inc. is prohibited.

- 12** **Mobile Environmental Control**
Dan Leland
- 20** **Motor Commutation Techniques**
Chuck Lewin
- 26** **Web-Implemented Irrigation System**
Chris Sontag
- 34** **Smart Building-Control Applications**
Beau Wadsworth
- 64**  **MicroSeries**
Digital Processing in an Analog World
Part 2: Technology Choices
David Tweed
- 72**  **From the Bench**
MIDI
Part 2: The Show Must Go On
Jeff Bachiochi
- 78**  **Silicon Update**
Socket Rocket
Tom Cantrell

- Task Manager** 2
Elizabeth Laurençot
Netting 100
- Reader I/O** 6
INK On-line
- New Product News** 8
edited by Harv Weiner
- Advertiser's Index/
December Preview** 95
- Priority Interrupt** 96
Steve Ciarcia
What's PC?

INSIDE ISSUE 100

EMBEDDED PC

- 40** **Nouveau PC**
edited by Harv Weiner
- 44** **All BIOSs are Not Created Equal**
Scott Lehrbaum
- 49** RPC **Real-Time PC**
Embedded RT-Linux
Part 1: General Introduction
Ingo Cyliax
- 57** APC **Applied PCs**
emWare Top to Bottom
Part 1: Monitoring via the Internet
Fred Eady

READER I/O

CONSIDERING THE ALTERNATIVE

I enjoyed Jeff's article "Transformerless Power Conversion" (INK 97), but he left out a method that performs more efficiently and costs less than what he described.

As sources of reactance, inductors generally perform poorly compared to capacitors. I appreciate the desire to eliminate bulky inductors from a power supply, but before abandoning reactance altogether, consider using capacitance.

A circuit with a series resistor (to resist fast line spikes) and a series film capacitor, followed by a bridge, a parallel Al-electrolytic capacitor, and a zener (five components) will perform well for a wide range of circuits. It offers higher efficiency (greater than 90%) and lower cost (less than \$0.50 in quantities over 1000). A \$0.20 250-V 0.47- μ F series film cap provides more than 8 mA. The current depends on the derivative of the line voltage and is hardly affected by the output voltage (1.5–36 V).

Joe Betts-LaCroix
Joe@analogdesign.com

Thanks for your comments. I'll have to plead guilty to a conscious omission. I cringe when I see capacitors being used this way. At any but the smallest currents, the heat build-up in these high-voltage capacitors accelerates component deterioration.

Yes, this method is cheaper, and I would encourage adding a MOV across the input to protect the capacitor from unforeseen over-voltages. However, from a safety perspective, I wouldn't base my product on it. In fact, I've heard rumors that this form of power-stealing circuitry will not be cutting the muter in future consumer devices.

Jeff Bachiochi

Editors note: The URL listed for Paradigm Systems in the sources section of Fred Eady's "Debugging & the Net186," (INK 97, p. 59) is incorrect. The correct URL is www.devtools.com.

November Design Forum password:

Control

INK ON-LINE

Your magazine enjoyment doesn't have to stop on the printed page. Visit *Circuit Cellar INK's* Design Forum each month for more great online technical columns and applications. Here are some of the great new on-line articles you'll see in November:

Columns

Silicon Update Online: Standard Chips Can't Be Counted Out—Tom Cantrell

Lessons from the Trenches: Bridging C and Assembly—George Martin

Forum Feature Articles

Build an 8051 Development System on a Budget—Bruce Reynolds

Creating a Flexible Custom Evaluation Tool Platform for ASICS—Eric Jacobsen

PIC Abstractions

Design Abstracts from our Design98 Contest

Digital Chromatic Tuner for Musical Instruments—

Darko Lazovic

The Video Capture Pro—Winston Gadsby

Dot/Bar-Graph Controller—Eduardo Sigal

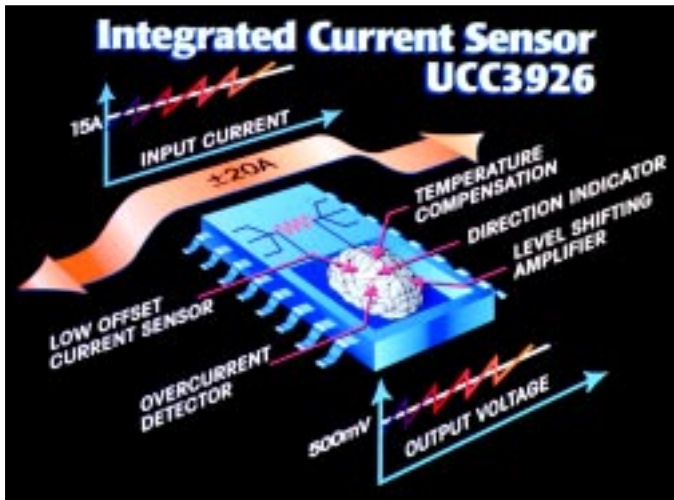
Missing the Circuit Cellar BBS?

Then don't forget to join the *Circuit Cellar INK* newsgroups! The cci newsserver is the engineer's place to be on-line for questions and advice on embedded control, announcements about the magazine, or to let us know your thoughts about *INK*. Just visit our home page for directions to become part of the newsgroup experience.

www.circuitcellar.com

NEW PRODUCT NEWS

Edited by Harv Weiner



CURRENT-SENSOR IC

The **UCC3926** current-sensor IC contains an internal current-sense element and precision amplification circuitry to offer a complete, single IC solution for current sensing. It uses an integrated noninductive current-sense element that eliminates the need for an external sense resistor, minimizes power loss, and offers higher efficiency. It handles input currents up to ± 20 A, and it addresses a wide

variety of power-management and industrial-control applications.

The IC contains a wide-band transimpedance amplifier for converting the current into a proportional voltage through an internal, noninductive, 1.3-M Ω shunt resistor. The sense element operates in both high-side (VDD referenced) or low-side (ground referenced) applications. A current polarity indicator (sign bit) that gives a positive or negative indication of the current flow direction is included.

A low-offset high-speed (90 ns) comparator provides over-current indication with user-programmable threshold for precise setting of the over-current alarm. Excellent linearity is made possible by temperature-compensated internal circuitry (± 200 PPM/ $^{\circ}$ C), eliminating temperature-related offsets including self-heating effects.

The surface-mount UCC3926DS is packaged in a 16-pin narrow plastic body SOIC and is priced at **\$3.60** in 1000-piece quantities.

Unitrode Corp.
(603) 424-2410
Fax: (603) 424-3460
www.unitrode.com

LOW-POWER CONTROL DEVICE

HulaPoint Remote combines an advanced motion algorithm with an innovative Hall effect-type sensor to provide accurate and effortless cursor control. The low-power device is designed for remote controls, such as those used for multimedia and game products, as well as telecom devices and H/PCs.

HulaPoint Remote provides either CMOS-level serial output or ASK modulated output, which is suitable for driving the external-power transistor and IRED (infrared emitting diode). Equipped with a wait mode, it consumes less than 50 μ A and operates in a 3–5-V range. It can be powered by three or four AA batteries and will operate for one year on three batteries.

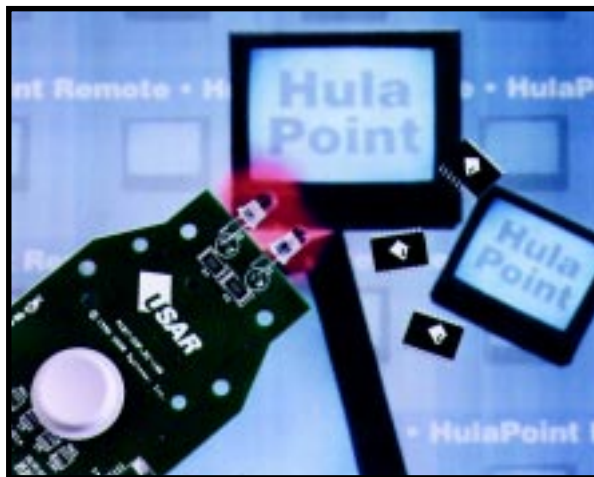
As well, HulaPoint Remote can encode up to 12 keys. These keys may be used for left, right, and middle mouse-button selection and as function keys. If

only the standard mouse buttons are used, HulaPoint is compatible with both the Logitech three-button and the Microsoft/IBM two-button mouse protocols.

The ergonomic sensor provides 10 $^{\circ}$ of movement in every direction. Because it uses magnet technology, there are no moving parts to break or wear out. This quality makes the device highly durable.

HulaPoint Remote is available as either a chip-and-sensor kit for integration on an existing PCB or as a drop-in module. Drop-in modules are designed to fit customer requirements.

An evaluation kit containing the remote and all the components necessary to test the device is priced at **\$150**.



USAR Systems, Inc.
(212) 226-2042
Fax: (212) 226-3215
www.usar.com

NEW PRODUCT NEWS

RUGGEDIZED VIDEO CAMERAS

The GBC **TKO-450** and **TKO-935C** (both known as The Brute) are discreet industrial-strength cameras that can be used indoors or outdoors. The heavy-duty die-cast camera case features a polycarbonate window, tamper-proof screws, and conduit fittings for cable entry, making the unit weather-proof and vandal resistant.

Both the **TKO-450** (B/W model) and the **TKO-935C** (color model) feature a 1/8" CCD camera that is internally adjustable on a three-way gimbal. An included corner mount features a ball joint for external adjustability. The camera comes standard with a clear polycarbonate window, and an optional gray window is available if concealment of the camera is desired.

The **TKO-450** has over 425 lines of resolution and a sensitivity of 0.03 lux. The **TKO-935C** has 330 lines of resolution and a sensitivity of 0.4 lux. The color model also has automatic through-the-lens white balance and backlight compensation for true-color rendition. Both models come standard with a 4-mm lens, electronic shutter, and power supply. Optional 2.5-, 6-, 8-, and 12-mm lenses are also available.

The **TKO-450** lists for **\$349.30** and the **TKO-935C** lists for **\$559.30**.

CCTV Corp.
(201) 489-9595
Fax: (201) 489-0111
www.gbc-cctv.com



8-BIT RISC MICROCONTROLLERS

The Microchip Technology **PIC12CE673** and **PIC12CE674** are one-time programmable (OTP) 8-bit RISC microcontrollers featuring EEPROM data memory, four-channel 8-bit ADC, and an on-chip oscillator. The '673 features 1024 × 14 words of OTP program memory, and the '674 offers 2048 × 14 words of OTP program memory. Both devices have 128 bytes of SRAM and 16 bytes of EEPROM data memory, enabling critical system-parameter storage to be distributed throughout the application. Typical applications include remote controls, security systems, battery chargers, PC peripherals, and smart sensors.

The devices have six I/O pins, 35 single-cycle instructions (400-ns execution time), 8-bit timer/counter with 8-bit programmable prescaler, and an on-chip 4-MHz clock oscillator for additional system cost and size savings. An in-circuit serial programming capability enables the microcontrollers to be programmed after being placed in a circuit board. Maximum power-consumption is 2.0 µA for maximum battery life.

The PICMaster Universal development system supports the microcontrollers. This Windows-based system features the MPLAB integrated development environment, which gives users the flexibility to edit, compile, emulate, and program parts all from a single user interface. The comprehensive PICMaster is available for **\$2490** without the Pro Mate II device programmer (**\$3345** with Pro Mate II). A CE-compliant version of PICMaster is available for European applications.

The **PIC12CE673** and **PIC12CE674** are available in 8-pin PDIP and windowed CERDIP packages. Pricing in 1000-unit quantities (4-MHz, commercial temperature version) is **\$2.04** each for the **PIC12CE673** and **\$2.39** each for the **PIC12CE674**.

Microchip Technology, Inc.
(602) 786-7200
Fax: (602) 899-9210
www.microchip.com

NEW PRODUCT NEWS

DIGITAL PANEL METER

The **DMS-30PC-4/20S** series of 4–20-mA current-loop input, 3.5-digit, LED display-panel meters consists of a loop-current conditioning board mounted to an epoxy-encapsulated voltmeter. These devices operate from a single +5-V power supply or an optional supply ranging from 7.5 to 32 V (24 V nominal). The 2.17" × 0.92" × 1.0" package features a large (0.56") red or green LED display and 100% soldered connections.

The advanced circuit design achieves a maximum loop drop of only 2 V. Two band-gap voltage references and precision metal film resistors ensure stable performance over the wide operating temperature range of 0° to +60°C. Gain (span) and offset (zero) adjustments are performed with high-precision, 20-turn potentiometers. All decimal-point and range-change settings are made on a gold-plated vibration-resistant DIP switch. There are no solder gaps or jumpers. Connections to the current loop and the power source are made on a rugged four-position screw-type terminal block.

A bezel assembly, featuring secure screw fasteners and an EPDM rubber gasket, is available for applications requiring moisture and dust resistance.

Low-power or high-intensity red LED models are also available. Prices for the meters start at **\$78** in single quantities.

Datel, Inc.
(508) 339-3000
Fax: (508) 339-6356
www.datel.com



NEW PRODUCT NEWS

DATA TERMINAL

The **QTERM-B30** pedestal-mount operator-interface/data terminal is a rugged low-cost device suitable for industrial applications. The terminal has a four-line \times 40-character supertwist lighted LCD, 55-key tactile membrane keypad with QWERTY-style layout, and four programmable indicator LEDs for operator feedback. The keypad legend and logo label can be fully customized. An optional manufacturer ID code means only terminals you buy operate with your equipment.

The QTERM-B30 comes with an EIA-232 serial interface. An optional EIA-422 serial interface is available. Data rates from 1200 to 19,200 bps are accommodated with a wide range of data formats. Approximately 50 software commands are available to the host for controlling the terminal, including cursor control, query commands, and hardware (lighting, buzzer, etc.) control.

The standard terminal uses an external 5-VDC supply, and an optional regulator enables operation with an external 7.5- to 24-VDC supply. The 8" \times 7.38" \times 1.5" unit is made of a rugged cast-aluminum housing with studs for pedestal mounting.

The QTERM-B30 sells for **\$585**.



QSI Corp.
(801) 466-8770 • Fax: (801) 466-8792
www.qsicorp.com

FEATURES

12

Mobile Environmental Control

20

Motor Commutation Techniques

26

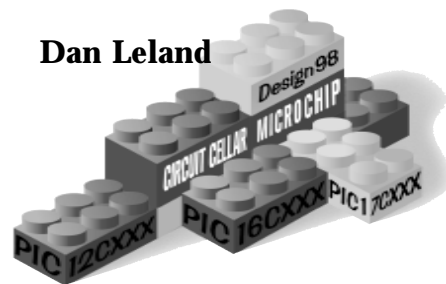
Web-Implemented Irrigation System

34

Smart Building-Control Applications

FEATURE ARTICLE

Dan Leland



Mobile Environmental Control

Access is power. Using a One-For-All universal remote, X-10 modules, and a PIC, Dan empowers the wheelchair-bound community, giving a quadriplegic friend remote access to the lights, thermostat, security system, and more.

a

friend of mine called me up a short time ago and asked if I could help him out. Apparently, he wasn't having much luck finding a device that would let him remotely control a few lights, his TV, stereo system, and possibly an electrical appliance or two. You might think it's nothing that a universal IR remote control and a little X-10 technology couldn't handle.

Well, that's not the whole story. There were a few other issues to think about. My friend is paralyzed from the neck down and is on a limited budget. His search of available assistive technologies turned up a fairly good selection of environmental control products, but only a few of the more expensive devices came close to meeting his needs.

In all fairness to the companies that develop special-needs technology, it's important to remember that people with severe physical disabilities represent a small market. A company that produces assistive devices by the dozens, or even hundreds, is not able to benefit from the kind of economies that typical consumer-electronics manufacturers enjoy. So, prices tend

to remain high for even the most basic technical aids.

Considering all the various commercialization hurdles facing manufacturers of assistive devices, I was somewhat surprised to find so few independently engineered adaptations of existing consumer products. Why couldn't a universal remote and a few X-10 modules be made a little more accessible?

NOT JUST ANY REMOTE

In many cases, the fact that most consumer electronics products are so highly integrated and miniaturized is probably the biggest barrier to creating accessible options. I have to admit, the idea of adapting an existing infrared remote wouldn't have seemed all that practical had I not been familiar with the One-For-All (OFA) series of universal remotes available from Universal Electronics.

Many of their devices are affordable, X-10 capable, and able to control a variety of audio-visual components. Add to the list a unique little feature

that enables you to operate an OFA without having to press the buttons, and presto, you have the makings of an accessible environmental-control solution.

Of course, I'm referring to the three-pin serial port built into the upgradable models, which was originally intended for in-house use as a means to update preprogrammed manufacturer codes.

Needless to say, it didn't take home-automation enthusiasts long to discover that all the OFA button functions can be accessed via the port. Connect a suitable host controller, and you have a fairly versatile gateway for home-control applications.

SUI AS IN GUI?

I wanted to avoid building a custom input device, so I tried to design an interface that could be used by people with physical disabilities other than quadriplegia. As a result, I decided to stick with some of the more tried and trusted methods that are used to control assistive devices. The answer I

came up with is a simple scanning type of controller.

The scanning user interface (SUI) has 40 labeled LEDs (arranged in a 4 × 10 array) corresponding to the buttons on the OFA infrared remote. As the name implies, a person scans the display and selects various control options using one or two input switches. The SUI translates the selection and sends the appropriate control command to the OFA through a serial connecting cable.

The SUI is built into a small black enclosure, occupying very little space (see Photo 1). It mounts next to the wheelchair armrest, where it's fairly inconspicuous yet visible to the user. The OFA mounts to the wheelchair frame, behind and above the user's right shoulder, for optimal IR beam spread.

TWO WAYS TO SCAN

A person navigates through the SUI display and makes selections using one of two operating modes. Auto-scan mode generates an automatic scanning sequence with three selectable speeds.

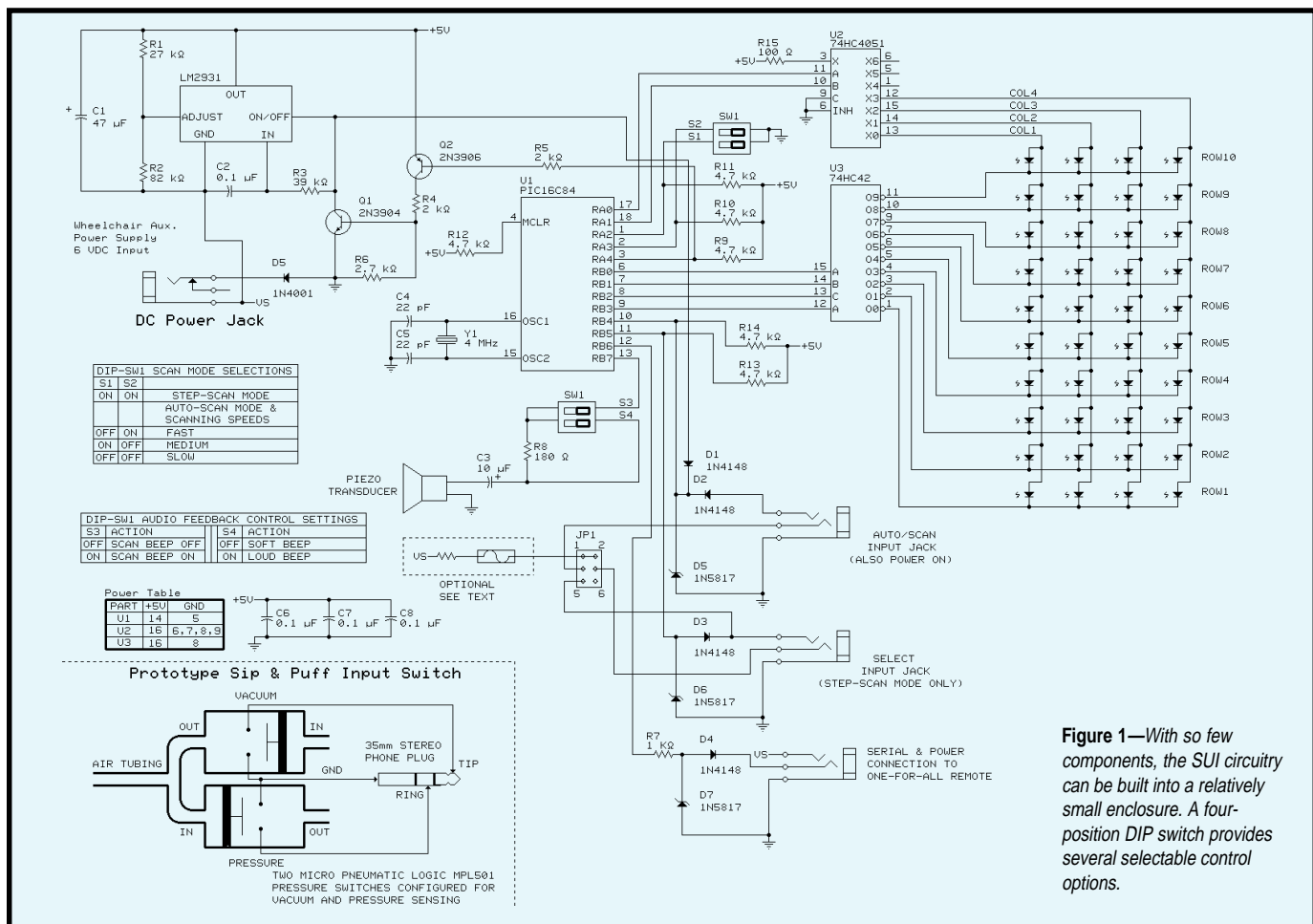


Figure 1—With so few components, the SUI circuitry can be built into a relatively small enclosure. A four-position DIP switch provides several selectable control options.

Both scan and select functions are controlled by a single switch. The first switch closure powers the SUI on, and the second starts the scan down the leftmost column of LEDs.

A third switch action redirects the scan across a desired row. Once the scan reaches the appropriate point, the final switch activation executes the OFA control function that corresponds to the labeled LED.

Auto-scan mode is designed to harness a single physical action, but it requires a certain amount of coordination to manage the auto-scan timing.

In step-scan mode, one switch controls the scan and a second performs the select function. The scan is incremented one step at a time with each activation of the scan switch.

The user reaches a desired selection, at his or her own speed, by alternating between scan and select switches. This provides greater control for those who have difficulty anticipating the timing of the auto-scan sequence. Step-scan is ideally suited for a sip-and-puff (pressure/vacuum) dual-action input switch.

The SUI accepts most ability switches or any SPST momentary contact switch. There's a wide range of ability switches available through manufacturers and distributors of assistive devices, and they're designed to respond to gross and fine physical movements (e.g., blink, finger-flex, head-tilt, sip-and-puff, and oversize button and paddle switches).

SUI INSIDE

I tried to keep the circuitry as simple as possible. The only frills in the bare-bones circuit of Figure 1 are a couple of setup DIP switches.

Aside from the PIC microcontroller, I used some easily available common parts. The PIC16C84 was an interesting choice, considering I had absolutely no experience with Microchip's product line. None of the microcontrollers I was familiar with seemed particularly suitable for this application, so I decided to try something new.

Aside from all the good things I'd heard about the 8-bit devices, I'm somewhat of a coward when it comes

to assembly programming. I knew there were some great software tools available for the PIC line of micros.

The '16C84 had just enough I/O for the job, and I didn't let a single pin go to waste. Almost half of the I/O-port pins are used to control the LED display array.

Although a display made up of discrete LEDs may seem a little low tech compared with an LCD panel, it's much easier to follow the scanning sequence when all the labeled selections are visible at once. Scrolling across a two- or four-line LCD to read multiple selections can get quite tiring after a while. An LED array, of course, becomes less practical as the number of selections increases.

Four Port B lines (RB0-RB3) and two Port A lines (RA0, RA1) are multiplexed by a 74HC42 decoder and a 74HC4051 analog switch to sink 10 LED rows and source four LED columns. Switches S1 and S2 of the four-position DIP switch (SW1) are read through pins RA2 and RA3 at powerup. The configuration of these switches puts the

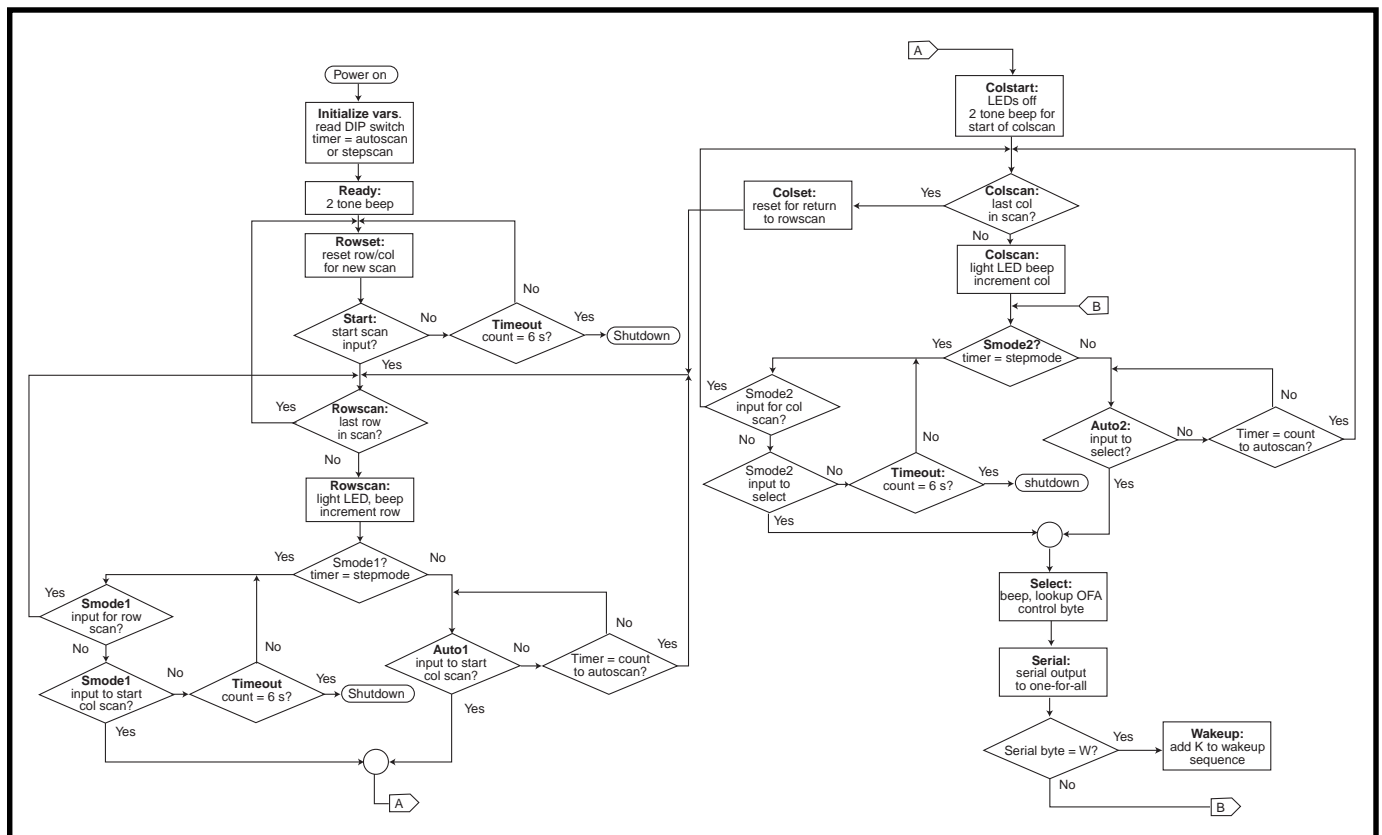


Figure 2—The SUI software begins with an initialization sequence and then waits for an input to start the scan down the rows. If the first step-scan or auto-scan mode input routines detect a select input, the program execution is passed to the column-scan segment. The column-scan input routines are similar, but a select input in either auto2 or smode2 results in serial-control commands being sent out to the OFA.

SUI into auto-scan or step-scan mode and sets one of three auto-scan speeds.

During scanning, RB7 outputs a PWM audio signal to a tiny piezo transducer through a simple on/off/attenuator circuit consisting of a resistor and switches S3 and S4 of SW1. The auditory feedback beep is enabled or disabled by S3, and two volume levels are provided by S4.

The OFA serial control data, output on pin RB6, and the DC supply voltage are routed out to the OFA through a 3.5-mm stereo phone jack.

The PIC detects scan-and-select user input through RB4 and RB5, respectively. Each I/O line is connected to the tip contact on separate 3.5-mm stereo jacks.

This setup enables the connection of individual off-the-shelf ability switches, which typically come with 3.5-mm mono plugs attached. Additional in-line diodes and grounded Schottky diodes provide a measure of ESD and negative-voltage protection on all I/O lines connected to external jacks.

The three conductor-input jacks and jumper JP1 provide several control-switch options for other potential applications. By jumpering the RB5 select line over to the ring contact on the auto-scan input jack, a dual-action sip-and-puff switch can be connected to the SUI via a single three-wire cable.

JP1 can also route the supply voltage out through the ring contacts on each jack. This aspect comes in handy when you need to use more specialized input-switch devices that require power for optical or magnetic sensor arrangements.

Although I didn't use this particular feature in my prototype, I recommend using a resistor or a resettable fuse on the voltage supply line to the header. Some kind of protective device is a good idea in case a mono plug is accidentally inserted into the stereo jack, connecting the power supply to ground.

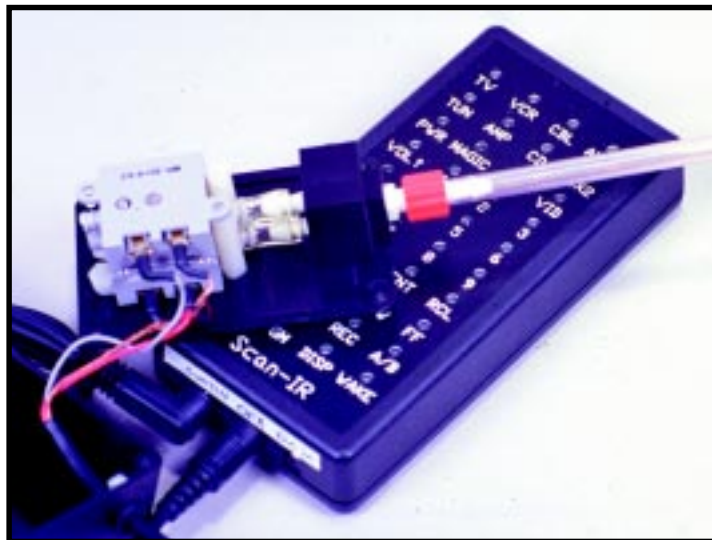


Photo 1—The scanning user interface, prototype sip-and-puff switch, and One-For-All 8 infrared remote (not shown) are the basic components for an accessible mini environmental-control system.

The tip lead on the auto-scan input jack is connected to the LDO regulator-enable line. When the tip-to-ground connection is initially made, the PIC software latches the regulator-enable line low, through pin RA4 and the PNP/NPN transistor combination.

Once powered, further tip-to-ground contact doesn't affect the regulator but is seen as valid auto/scan input by RB4. If no user activity is detected on RB4 within approximately 6 s, the software sets RA4 high and shuts down the regulator to help minimize power consumption.

The 6-V supply is slightly reduced by a polarity-protection diode installed on the ground terminal of the DC power jack. The LDO regulator handles the lowered input voltage quite nicely and supplies 5 V to the circuit.

The supply voltage is passed straight through to the OFA, so the input should never exceed 7 V. The SUI and OFA combination draws a maximum of ~55 mA when the OFA transmits an infrared command.

SUI ONBOARD

Although my friend has good head movement and slight control of his fingers, he prefers to use a sip-and-puff interface to operate his electric wheelchair and access his home computer. I made the mistake of building a sip-and-puff switch for his SUI before I looked at the electronics on his wheelchair.

It turned out that the wheelchair sip-and-puff controller could be cycled between drive functions, seat adjustments, and several auxiliary relay outputs. A short connecting cable between the SUI and two of the relay outputs eliminated the need for an additional switch.

Optional control outputs aren't necessarily common to all powered chairs, so an additional input device is still required in some cases. The schematic for a prototype sip-and-puff switch using two Micro Pneumatic Logic (MPL) dual-port pressure switches appears at the bottom right of Figure 1.

Pressure and vacuum sensing was obtained by simply teeing the air-supply tube to the input port of the puff switch and to the output port of the sip switch. The MPL 501s have diaphragm-operated contacts that respond to pressures ranging from 0.1" to 0.75" of water.

A very small amount of the sensitivity can be adjusted with a setscrew that changes the contact spacing. I found these switches to be a little too sensitive, so I'd probably try a higher rating for any future applications. The pressure-switch terminals are wired with a common ground to a single three-conductor cable with a 3.5-mm stereo plug attached.

As luck would have it, I was able to get power for the SUI from an auxiliary outlet on the wheelchair that was clearly intended for some other type of accessory. I contacted the local dealer's service department and obtained all the necessary technical details before attempting the SUI connection.

The output was well isolated from the rest of the controller circuitry and provided a clean 6 VDC at 750 mA. Like the auxiliary control relays, some wheelchairs may have a similar power option and some may not. An alternative solution is to tap the wheelchair batteries directly with a small DC-to-DC converter that isolates the SUI and OFA from any motor-generated transients.

BACK TO BASICS

As this was my first PIC project, I wasn't keen on spending too much time or money on software development. I needed something to get me up and running as quickly as possible, so I ended up purchasing the PicBasic Compiler and Epic Programmer from microEngineering Labs.

PicBasic commands are quite powerful, and it doesn't take long to string a few together to get things going. `Button`, `Sound`, `Pins`, and `Serial` are just a few examples of I/O commands that handle input, audio feedback, display control, and serial communications for the SUI program.

Typically, a pin assignment and a couple parameters for each command are all you need. Although Port B pins are the only ones supported by these PicBasic commands, Port A pins can be accessed with `Peek` and `Poke` calls.

The program code, like the hardware, is fairly straightforward. As you see in Figure 2, there are four basic program segments—initialization, row scan, column scan, and serial output. The program starts by setting all the variables and I/O ports, which effectively turns off the LED display and enables the regulator through pin RA4.

Port A is read to obtain one of four values for the DIP-switch settings on RA2 and RA3. This value is adjusted and assigned to the `Timer` variable. Three values determine the auto-scan speed. The fourth sets a branch condition further on in the program, which runs the step-mode rather than the auto-scan input routine.

After the initialization, a dual-tone beep signals the SUI is ready for input. The `Row` and `Column` display control variables are reset for a new scan sequence. An input loop polls RB4 for a start-row-scan signal while incrementing a loop-pass counter. If no input is detected before the counter reaches a timeout limit (~6 s), pin RA4 is set high and the regulator is shut down.

If input is detected, the `Row` and `Column` values are output on Ports A and B, lighting the LED in the first row of the first column. A corresponding beep is generated, and the `Row` variable is incremented in advance for the next pass. At this point, the `Timer` variable

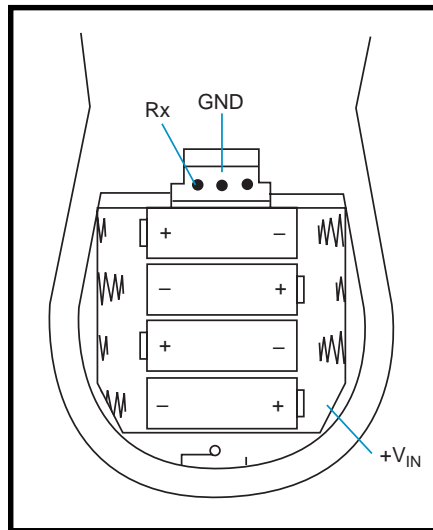


Figure 3—Removal of the OFA8 battery cover reveals the contact points for the installation of the SUI connecting cable.

is tested and the program branches to the step-mode input routine or continues in auto-scan mode.

The auto-scan 1 input routine polls RB4 again, but this time the loop-pass counter limit is set by the `Timer` variable. If no input is detected before the count reaches `Timer`, the program jumps back, increments the scan, and returns to the input loop.

Without input, this cycle continues, resulting in an automatic scanning sequence down the left column, with the `Timer` value setting the speed. Any input during the scanning process is seen as a `Row` selection, and program execution is passed over to the column-scan routines. When the last row is reached, the program goes back, resets the display variables, and waits for another start-row-scan signal.

The step-mode 1 input routine polls RB4 to increment the scan and then RB5 to select a row. If input is detected on RB4, the program jumps back, advances the scan, and returns to the input loop. If no input occurs on RB4, then RB5 is polled for a `Row` select condition. If no input is detected on either pin, the loop repeats until the 6-s timeout counter eventually shuts down the regulator.

If a `Row` selection is detected in auto-scan 1 or step-scan 1, the scanning sequence is temporarily halted, the program jumps to the column-scan handling routines, and a dual-tone beep signals the change.

The code structure is essentially the same for column scanning as it is for rows. This time, however, it's the `Column` variable that gets incremented, resulting in a left-to-right scan sequence across a selected row.

When the last column in a row is reached, the program resets the variables and returns to the row-scanning sequence down the left column. The `auto2` and `step-scan2` input routines poll RB4 and RB5, but instead of redirecting the scan, the input invokes the OFA serial control routines.

A beep indicates an OFA control selection has occurred. The program derives a value of 0-39 from the current `Row` and `Column` variables and assigns it to the variable `Key`. A look-up command uses `Key` as an index variable to retrieve a value from a table of 40 numeric constants. Each constant in the list corresponds to an OFA serial button command.

The serial-out subroutine sends the ASCII-character equivalent of the byte constant out to the OFA. If the byte equals an ASCII "W", the OFA wake-up routine kicks in and sends a "K" to complete the wake-up sequence that puts the OFA in serial-command mode.

The program returns to the last input routines and checks for further selections, which enables consecutive button presses for repeating functions like volume up. If no further input is detected, the program eventually returns to a point where a timeout routine powers the SUI off.

THE OFA CONNECTION

The One-For-All 8 remote I used in this application is connected to the SUI with a 36" audio patch cable with molded 3.5-mm stereo phone plugs on each end. One of the plugs is removed and the leads are soldered to the three points located beneath the OFA battery compartment (see Figure 3).

With reference to the stereo-plug contacts, the tip lead is soldered to the +V_{in} battery spring contact, the ring lead is soldered to the Rx pin, and the ground lead is soldered to the GND pin. Finally, the side of the battery cover is notched to accommodate the cable, and it is then snapped back into place.

OFA WAKE-UP CALL

Before the SUI can send over commands, the OFA8 must be initialized for serial communications. First, one of the OFA8 keypad buttons must be manually pressed to bring the OFA8 micro out of sleep mode for about 5 s.

The SUI is then activated and the Wake function is selected before the micro goes back to sleep. This serial wake-up command suspends the OFA sleep-mode routine and keeps the OFA in serial mode. The OFA8 returns to manual mode if the power is removed, so this procedure must be done the first time the unit is powered up or after any power interruptions.

The SUI auto-power-down feature helps offset the fact that the OFA is always on in serial mode. The manual keypad press in this set-up sequence is a bit of a drag, but it's specific to only four OFA models that I know of, including the OFA8.

The older OFA12 and OFA6 can be switched to serial-control mode with just a serial wake-up command, and they don't require the physical button press. Unfortunately, the OFA12 is discontinued and the OFA6 lacked the program capacity for my friend's application. The OFA6 is still readily available and probably meets the needs of most people.

With the OFA in serial mode, the SUI can send over serial-control commands corresponding to any of the OFA button functions. By following the instructions outlined in the One-For-All 8 user guide and code book, you can access all the features of the OFA through the SUI.

If the user needs help with the OFA setup procedure, manufacturer codes or macro sequences can be entered manually anytime after the OFA is powered on and before the SUI sends the wake-up signal. The OFA retains programmed information even if the power supply is temporarily removed.

SUM OF THE PARTS

So, how much accessibility does \$75 worth of parts buy? In my friend's case, a fair amount.

From his wheelchair, he now has control over ten lights, two sets of drapes, a ceiling fan, thermostat, secu-

urity system, a door actuator, and his home entertainment system. Even with the added cost of various X-10 accessories, he still had enough left over to take me out for coffee.

Unfortunately, that's when he gave me his list of suggestions for Version 2. Hmm...let's see: telephone control, EEG interface, robotic assistant...? 📧

Dan Leland has spent the past 11 years working in research and development for the Neil Squire Foundation, a Canadian nonprofit organization whose goal is to provide innovative services and technology to individuals with significant physical disabilities. For more information on some of the Foundation's activities, please visit www.neilsquire.ca. You may reach Dan at rd@mindlink.bc.ca.

SOURCES

PIC16C84

Microchip Technology, Inc.
(602) 786-7200
Fax: (602) 899-9210
www.microchip.com

MPL501 pressure switches

Micro Pneumatic Logic, Inc.
(954) 973-6166
Fax: (954) 973-6339
www.pressureswitch.com

One-For-All 8

Universal Electronics, Inc.
(330) 487-1110
Fax: (330) 487-1131
www.ueic.com

PicBasic compiler software, EPIC programmer

microEngineering Labs, Inc.
(719) 520-5323
Fax: (719) 520-1867
www.melabs.com

X-10 control accessories

X-10 USA
(800) 675-3044
(201) 784-9700
Fax: (201) 784-9464
www.X-10.com

Home Automation Systems, Inc.
(800) 762-7846
(949) 221-9200
Fax: (949) 221-9240
www.smarthome.com

FEATURE ARTICLE

Chuck Lewin

Motor Commutation Techniques

It's a never-ending race to get the most performance from your system. But, according to Chuck, motor commutation is a good springboard for flying over the hurdles of oscillations, overshoot, and all those other motor-stability problems.



Machine designers have long sought to get the maximum performance from their systems. But as they push throughput to the limit, they often fight a battle with motion-stability problems like torque ripple, oscillation, or overshoot.

Motor commutation is an important weapon in this battle. Recently, there has been an increase in the number of applications using software-based commutation performed by high-speed DSP chips. By doing commutation in software, it's possible to improve the motion's smoothness and accuracy. It may also reduce the audible noise level of the motor during operation.

Historically, commutation was performed using hardware-based Hall-effect sensors. This scheme requires commutation sensors in the motor, as well as electronics in the motor amplifier, to perform the phasing.

By taking advantage of the power and flexibility of software-based systems, it becomes possible to perform commutation functions in the controller's CPU. This software-based method uses the position feedback device of the motor (typically an in-

cremental encoder) to generate a sinusoidal commutation waveform.

Software-based commutation also performs advanced functions like automatic phase finding and waveform shaping. These two techniques let you eliminate Hall-effect sensors and also permit the motor performance to be improved by linearizing the commutation output tuned to a specific motor.

In this article, I explore the capabilities and benefits of software-based sinusoidal commutation and show how it can be used in applications involving brushless motors. I also examine some other commutation schemes and offer practical guidelines on which scheme works best in a given application.

SINUSOIDAL COMMUTATION

The term "sinusoidal commutation" refers to the waveform of the motor currents used to drive the brushless motor windings. Figure 1 illustrates these waveforms for two commutation phasings—a 120° phasing scheme used with three-phase brushless motors and a 90° phasing scheme used with two-phase brushless motors.

Sinusoidal commutation has gained popularity lately because it has the potential to operate a given motor more smoothly. It also offers more predictable behavior than the same motor commutated using traditional six-step techniques.

Although sinusoidal commutation has been used in a wide variety of applications, it's especially popular in systems that are sensitive to torque disturbances. Linear motors and other direct-drive brushless applications fall into this category, as do applications like medical automation, semiconduc-

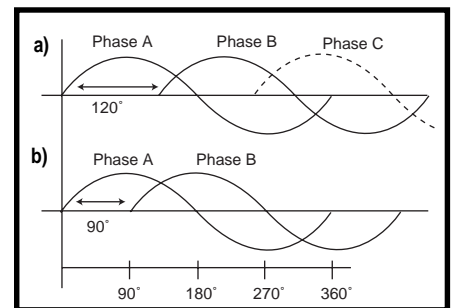


Figure 1a—Commutation waveforms for three-phase permanent magnet brushless motors are separated by 120°. b—Waveforms for two-phase brushless motors are separated by 90°.

tor automation, scientific instruments, and analytical equipment, among others.

MOTOR-OUTPUT SIGNAL GENERATION

In a digital sinusoidal commutation scheme, the motor-output signal is calculated from the position encoder as well as from information about the initial phasing of the motor windings. The output signal to each winding is determined digitally by synthesizing a sinusoidal waveform and combining it with the overall desired torque output of the servo filter.

Figure 2 shows the control flow for a typical digital controller that synthesizes the sinusoidal waveforms using an internal ROM look-up technique.

Typical digital commutation schemes read the encoder position on a discrete time-sampled basis and determine a new motor output value for each of the three brushless motor phases. State-of-the-art systems perform commutation waveform lookup at least at 10 kHz, and many perform at 15+ kHz. For high-speed applications, such as spindle motors running at 30,000+ rpm, accurate high-frequency waveform synthesis is a must.

Given the current shaft angle (encoder position) and initial phasing of the motor (which locates the shaft to the proper electromagnetic position), you can calculate the desired current in each motor coil for a phase angle for three-phase brushless motor via:

$$\begin{aligned} \text{phase_A_angle} &= (\text{Encoder} - \text{init_phase} \\ &\quad \% n_counts) \times \frac{360}{n_counts} \\ \text{phase_B_angle} &= \text{phase_A_angle} - 120 \\ \text{phase_C_angle} &= \text{phase_A_angle} - 240 \end{aligned}$$

and the value output on each phase signal is:

$$\begin{aligned} \text{phase_A_output} &= \sin(\text{phase_A_angle}) \times \text{torque} \\ \text{phase_B_output} &= \sin(\text{phase_B_angle}) \times \text{torque} \\ \text{phase_C_output} &= \sin(\text{phase_C_angle}) \times \text{torque} \end{aligned}$$

where *Encoder* is the current encoder location in counts, % is the modulo operator, *n_counts* is the number of

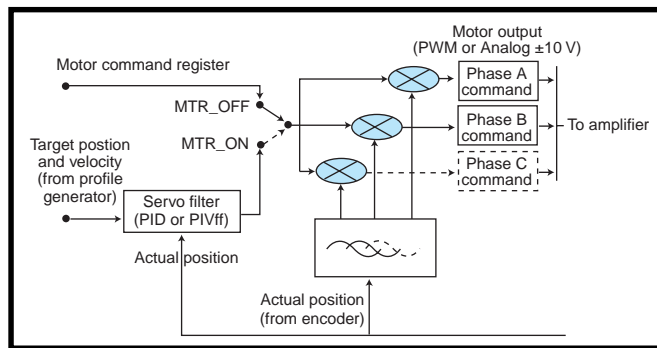


Figure 2—Here's the control flow for a typical commutating digital controller. The torque command is provided manually (motor command register) or by position servo (PID). Then, the torque command is vectorized into A, B, and C commands and set to the amplifier using encoder feedback to determine the phase angle.

encoder counts per electrical cycle, and *init_phase* is the initial phase angle offset in encoder counts.

MAKING CONNECTIONS

Figure 3a shows typical connections for an all-digital sinusoidal commutation controller, like those available from motion-card vendors. The digital controller provides two analog output signals (using a DAC): phase A and B.

An amplifier circuit (on the same PC board or on a separate board) generates the phase C signal using the equation $C = -(A + B)$. This amplification technique is used with a current-loop scheme that converts the analog voltages output by the motion processor into torque commands to the motor.

Figure 3b shows a digital controller using a direct PWM output technique so three output signals (phase A, B, and C) are provided. These signals are sent directly to H-bridge-type amplifiers to drive the motors.

These amplifiers may be located on the board or in a separate amplifier. This technique is used without a current-loop circuit, so that the motor is controlled in voltage mode.

PHASE INITIALIZATION

An important question concerning the adoption of software-based sinusoidal commutation is how the motor phases are initialized after powerup. Position feedback is usually provided by an incremental encoder. On powerup, no absolute information is available concerning the correct motor phasing.

This situation presents a serious challenge to the universal adoption of encoder-based sinusoidal commutation. To address this challenge, most digital controllers support two distinct phase initialization schemes—the first using no hardware sensors at all other than the encoder itself, and the second using the Hall-effect sensors—in conjunction with the encoder.

ALGORITHMIC INITIALIZATION

The first method, often called algorithmic initialization, performs a short phase excitation procedure that applies power to the motor windings and observes the motor response. Based on the settling position of the motor, the correct commutation can be determined. This technique provides accurate results and doesn't require separate commutation sensors like Hall-effect sensors.

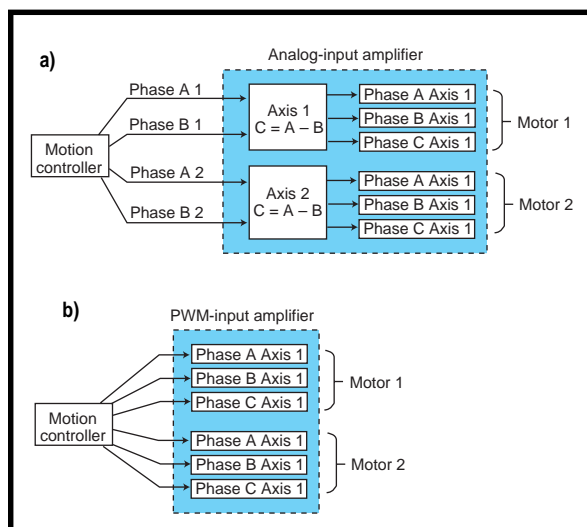


Figure 3a—A controller with onboard commutation in analog output mode sends two phase signals (A and B) to the amplifier. The amplifier constructs the third phase (C) using the current loop to ensure that the sum of the phases is 0. **b**—A controller with onboard commutation in PWM output mode sends three phase signals (A, B, and C) to the amplifier. Each signal drives the amplifier's half bridges directly. With PWM input, the amplifier usually runs in voltage mode and doesn't perform a current loop.

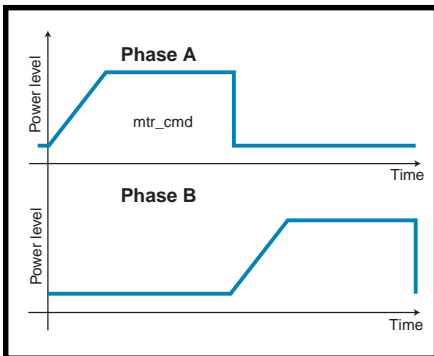


Figure 4—This typical excitation sequence of motor windings to determine initial phasing uses algorithmic techniques. The phase-A current increases, holds, and falls off. Current B goes through the same sequence. The rotor angle just before the phase-B current angle determines phasing.

The disadvantage is that, while power is being applied, the motor bumps uncontrollably. And if the load friction is high or even against a hard stop, the procedure does not operate correctly at all. But, most software-based controllers recognize this condition because the motor does not move.

Figure 4 shows a phase excitation sequence used with algorithmic phase initialization. More complicated sequences that try to minimize the motor movement may be adopted, but these techniques require the motor to move at least a small distance.

Algorithmic initialization techniques that don't require the motor to move at all have been explored but are not yet in common use. These techniques typically attempt to measure the main vector angle axis of the inducted winding voltage when excited via a fixed frequency of all three windings.

HALL-BASED INITIALIZATION

In the second phase-initialization method, three Hall-effect sensor signals for each axis are connected directly to the digital controller and are used just during phase initialization.

The advantage is that the motor phasing can be determined from the moment the motor is powered on. Once the Hall-sensor information has been collected during initialization, only the encoder is used and the motor is commutated sinusoidally.

WAVEFORM SHAPING

Sinusoidal commutation is generally an improvement on square-wave or

Hall-based commutation not only because of the elimination of torque discontinuities but also because the motor torque resulting from sinusoidal commutation is more linear than the torque resulting from six-step.

The sinusoidal waveform more closely matches the optimum waveform required by the motor to create perfectly linear torque output.

But, motors specifically purchased to be commutated using sinusoidal commutation do not create perfectly linear torque output. Figure 5 shows typical torque output for a motor that is commutated using sinusoidal commutation at a fixed torque level.

As the graph indicates, the actual torque output deviates from linear by 5–10%. This deviation represents the torque ripple that can be expected from sinusoidal commutation. Six-step commutation torque ripple tends to be in the 10–15% range.

Waveform shaping is occasionally used when the motor characteristics are consistent and well understood and when this torque ripple is not acceptable. The basic technique is to transform the sinusoidal waveform by multiplying by a compensating torque magnitude, thereby linearizing the resultant torque output. Figure 6 shows this, dramatically exaggerating the magnitude of the correction.

Although this technique is appealing and easy to implement, it has several limitations. First, the torque response of a given motor may vary as much by

the type of motor, as by each physical manufactured motor. So, it's important to understand the torque variations over a sample of specific motors to develop an optimum average waveform.

Also, the torque ripple magnitude and characteristics may not be proportional to the commanded motor torque level. As an effect, a compensating waveform, which is optimal for one motor torque level, may not be optimal at a lower or higher level.

Detent torque is an example of such nonlinearity. Detent torque is a torque disturbance found in some types of brushless PM motors. It results from the magnetic attraction of the magnets in the rotor to the stator at specific rotation points.

Detent torque is typically not proportional to the commanded motor torque operating level, so it can't be completely removed this way. However, it may be possible to lower detent torque by modifying the waveform in other ways—for example, a fixed output offset for each motor position.

OTHER COMMUTATION CHOICES

Sinusoidal commutation as a method for driving brushless DC motors has been gaining popularity in recent years because it offers smoother motion with no discontinuities in the power applied to the motor.

Several other commutation techniques are commonly used, each with a particular domain of applicability. Understanding the strengths and

Method	Advantages	Disadvantages
Hall-effect six step	Widely used	Not as smooth because of discontinuous application of power through each motor coil Hall sensors required
Digitally synthesized sinusoidal with Hall-based initialization	Smooth motion; high performance	Motor movement during phase initialization In typical applications, resolvers and associated R/D circuit is more expensive than the incremental encoder
Digitally synthesized sinusoidal with algorithmic initialization	No sensors beyond encoder required	
Resolver-based sinusoidal	Commutation circuitry is fairly simple	
Linear Hall-based sinusoidal	Commutation circuitry is fairly simple	Output of Hall sensors is not truly sinusoidal; not commonly available
Sensorless (back-EMF)	Very low cost; no sensors required	Will not work at DC; not appropriate for positioning applications

Table 1—A third method (internal to the motor) also uses Hall sensors but simply packages them on a small circuit board mounted inside the motor.

weaknesses of each scheme enables designers to optimize their choice of commutation techniques.

HALL-BASED COMMUTATION

Hall-based (a.k.a. trapezoidal or six-step) commutation is the most common method of commutating brushless motors in positioning applications. Figure 7 shows the relationship of typical Hall-sensor signals to the resulting current flow through each motor coil.

Each Hall sensor outputs a binary high or low value, switched every 180 electrical degrees. The three Hall sensors may be phased relative to each other in various ways, but in all cases, the sensors encode six unique states per electrical cycle with a resulting drive waveform as shown in Figure 7.

The advantage of this scheme is its popularity, as well as its low cost and ability to operate from DC (motor standstill). A disadvantage is the discontinuous current applied through each coil. At the Hall switching points, positioning systems may experience servo instability or torque discontinuities because of the abrupt changes in the power applied to each motor coil.

RESOLVER-BASED COMMUTATION

Resolvers provide sinusoidal information about a motor's shaft position which is electronically extrapolated using an IC known as an R/D (resolver to digital) converter to generate a digital representation of the motor shaft angle.

One benefit of the sinusoidal signal output by a resolver is that it can be used to commutate the motor. An analog-multiplier IC generates three phased signals by electronically multiplying the unphased torque signal from the controller with the three phased resolver feedback signals. This technique is nearly identical to the all-digital one except that it's performed in analog.

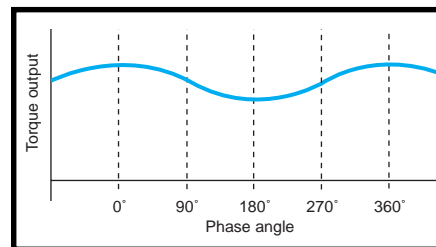


Figure 5—Even if perfect sine-wave current commands are sent to each motor phase, the motor's torque output may not be linear. The output may be distorted by the geometry, mechanics, and materials of the motor.

The advantages of this technique are that it is cost effective and accurate. Additionally, the resolver requires no phase-initialization procedure.

Its disadvantages are that it is not fully digital and it depends on analog circuitry, which may drift over. As well, off-the-shelf resolvers and their associated R/D converter chips are expensive compared with an incremental encoder of comparable resolution.

LINEAR HALL SENSORS

Another technique uses special Hall sensors that provide three sinusoidal output signals that can be used to phase the motor command using analog multiplier circuits. The technique that generates the final motor commands through each coil is identical to resolver-based feedback.

This technique has the advantage that it does not require an initialization procedure. On the other hand, the analog waveforms output by the Hall sensors are seldom truly sinusoidal. This situation may distort the torque applied to each coil, resulting in increased motor ripple. Also, these sensors are not commonly available for most motors.

SENSORLESS COMMUTATION

Sensorless commutation techniques use information generated by the motor during operation to determine motor phasing. The most common signal is

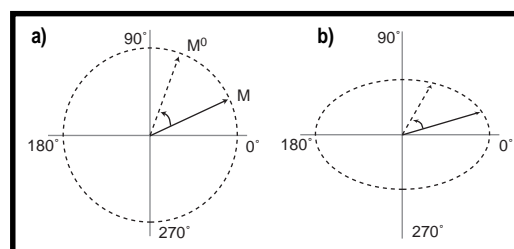


Figure 6a—A pure sinusoidal waveform has constant magnitude over the entire commutation electrical cycle, so that M equals M^0 at all times. **b**—The shaped waveform has variable magnitude so M does not equal M^0 at all phase angles. The oval modification is shown here. More complicated modifications may be required based on motor behavior.

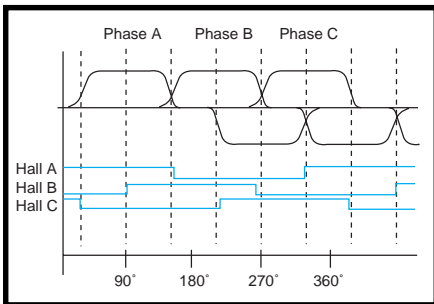


Figure 7—The typical phase excitation sequence for Hall effect-based commutation is shown here in six steps. Three Hall sensors determine the phase output at any given phase angle.

back-EMF voltage, which is created in the coil as the permanent magnet of the brushless DC motors rotates past it.

Presently, sensorless commutation techniques can only be used when the brushless motor is rotating, and therein lies its greatest limitation. This technique cannot be used for positioning applications because the commutation information disappears at DC when the motor stops moving.

However, it requires no sensors of any kind and is commonly used in high-volume applications like disk-drive

spindle control and fan control. For these applications, try a low-cost IC that implements sensorless control.

IT'S YOUR CHOICE

I examined the characteristics of a software-based implementation of sinusoidal commutation. Because of the broad applicability of sinusoidal commutation to brushless PM motors, it has substantial merit for high-performance motion systems.

Given lowering electronics prices, the cost of sinusoidal commutation for systems that already use a position encoder has become about the same as for less advanced six-step schemes.

Despite the strengths of software-based sinusoidal commutation, there are several alternative schemes that are worth considering. For a given application, system cost, desired performance, and sensitivity to disturbances, among other factors, play an important role. Table 1 summarizes the techniques I discussed.

Which commutation technique will work best in your application?

That depends on many factors that only you can judge. 📧

Chuck Lewin is chief technology officer of Performance Motion Devices. He has been working in motion control for eight years and designing DSP-based motion systems for five years. He has written articles for various engineering magazines, providing practical, application-oriented advice on the implementation of motion-control systems. You may reach Chuck at lewin@pmdcorp.com.

SOURCES

MC1231A

Performance Motion Devices, Inc.
(781) 674-9860
Fax: (781) 674-9861
www.pmdcorp.com

5651 motion-control card

Technology 80, Inc.
(800) 545-2980
(612) 542-9545
Fax: (612) 542-9785
www.tech80.com

FEATURE ARTICLE

Chris Sontag

Web-Implemented Irrigation System

Tired of relying on the weatherman in this El Niño year? Chris is, too. After seeing all those automatic sprinklers working in the rain, he decided to hook up a controller to the 'Net for live updates, making this device truly intelligent.



You're driving in a rainstorm that's been going all day and pass a house or business with its automatic sprinklers watering the lawn. So much for water conservation (a perennial topic of discussion during non-El Niño years in the arid desert climes of the Western U.S.).

No doubt, the building's owner set the automatic sprinkler timer to start watering the lawn on a preset schedule, and the microcontroller in the sprinkler control unit obediently turned the sprinklers on—rain or shine. The owner is either too far away to change the device or the process of changing the program at the device is too complex.

Somewhere along the line, we started calling even the simplest electronic devices "intelligent." But if electronic devices are really intelligent, why do they water the lawn when it's raining? These devices are only as smart as the information they're given.

A truly intelligent device should accept dynamic information and make adjustments based on that information. It may even take forecast information and weigh decisions based on what is likely to happen in the future.

You might argue that this ability takes far too much capacity to build into simple devices like a sprinkler timer with an 8-bit microcontroller.

But, adding a 32- or 64-bit micro would drive the price out of range for the consumer market. So "intelligent" devices continue to be unintelligent, and we run to the sprinkler controller whenever it rains.

But what if you could connect a simple device to dynamic information without adding resources to the device? A sprinkler controller that monitors environmental conditions or forecast information and changes its actions based on that information would save money and be politically correct. The device would be educated with updated information, making it truly intelligent.

INTELLIGENT ARCHITECTURE

To make a sprinkler system intelligent, you need three things. You need an interface to enable easy access and control of the device from anywhere.

You also require a way to gather and analyze current environmental conditions as well as a decision-making process to adjust the device based on current information. Also, a solution based on Internet standards makes the interface user friendly and easy for the device manufacturer to implement.

Using emWare's EMIT (Embedded Micro Internetworking Technology) software, I can put an intelligent system into place. EMIT includes five modular software components:

- emMicro, a compact, special-purpose micro web server that requires less than 1 KB of device memory
- emNet, a message-based protocol that combines packet and stream interchange
- Microtags, preprogrammed packets that define device controls (e.g., switches, buttons, LEDs)
- emGateway, which expands Microtags into their full parameters
- emObjects, a library of JavaBean components consisting of visual and utility objects

These components work together to create a dynamic user interface without requiring extensive resources from the device itself. Figure 1 shows how these components work together.

EMIT software is placed in three areas—on the device, in a gateway

browser, and at a user interface. The device is embedded with emMicro, Microtags, and emNet, as well as variables, functions, events, and documents.

The user interface is a standard web browser, enabling access via a Java-enabled GUI. The browser communicates with the gateway, which sends information to and pulls information from the device.

Between them lies the key to networking small devices to the Internet. Using emGateway minimizes resource requirements at the device by moving the workload to the client side of the equation.

When a user requests the device interface from a web browser, the browser sends the request to emGateway, which translates the high-level request and sends it to emMicro at the device. emGateway receives information through the Microtags built into the device, and substitutes each Microtag with a corresponding emObject.

emObjects are JPEG or GIF images or Java applets that represent device controls to the browser. The page with the substituted emObjects is then sent to the browser.

EMIT components process user requests to view and set device information, dynamically representing the results on the desktop. For example, users may ask to turn a sprinkler valve

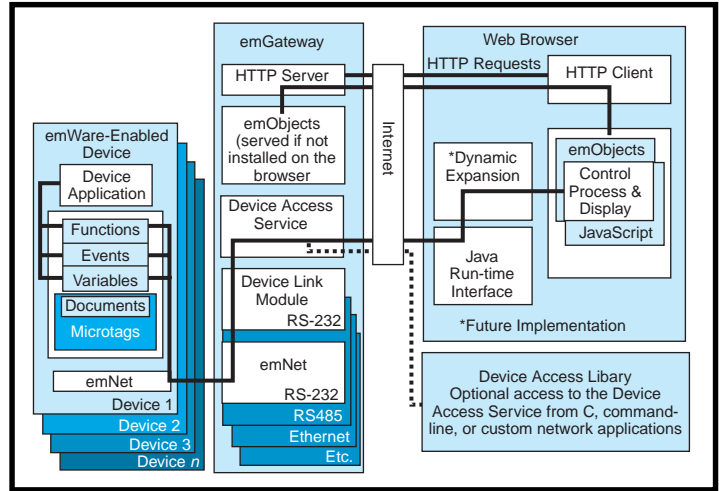


Figure 1—The EMIT distributed architecture moves the majority of resources away from the device, providing full Internet networking for devices with microcontrollers as small as 8 and 16 bits. The flexible architecture, with emGateway as the key, meets a variety of solutions and implementation needs.

from off to on. The browser then invokes emGateway, which sends the request to emMicro at the device.

When emMicro receives the request, it causes the device's microcontroller to flip the valve switch. Because device information has changed, the state of the embedded device changes, so a state-change message is sent to any interface emObject (component) monitoring affected variables. The interface now shows the valve state as on.

In addition to the 1-KB emMicro web server, this application is about 900 bytes. If you have to build physical interface logic onto the device (LCD, buttons, switches, etc.), you need another 2–4 KB of program space.

For applications on devices with larger processor capacity, the emGateway requirements can be moved to

the device. But for this example, you can minimize device requirements to reduce expenses.

ADDING SMARTS

The EMIT components enable control of the system through a GUI from anywhere at any time. Now we need to add information to educate the device regularly, allowing the sprinkler system to water proportionally—less in the spring and more in the heat of summer.

Because the device has Internet connectivity built in, it can check the

National Weather Service's web site for precipitation information and forecasts. For more accurate information, you can connect the system to temperature sensors, moisture probes, and wind monitors at the property.

Using the forecasts, you can program the device to water in the early morning hours of hot, dry days, providing the necessary moisture to the lawn and avoiding heavy evaporation that comes from watering in the heat of the day. On days with rain in the forecast, the sprinklers could automatically shut off.

When your computer is off or the sprinkler system is unable to access the Internet or weather-station information, the system simply defaults to its regular schedule or to user input.

You can even program the device to weigh current data (temperature, humidity, rainfall, etc.) against forecasts. For example, if the day is hot and dry, but the forecast calls for a heavy storm at night, the device can postpone watering and let the coming storm take care of the lawn. On days when the predictions are wrong, you simply call up the sprinkler interface and reset the sprinklers.

The sprinkler interface allows for customized usage in other ways. For example, you can turn off the system for a few hours to accommodate an unexpected thunderstorm or turn the system on if an expected storm fizzles out. On the other hand, if you're away from home (or forgetful), the sprinkler

Listing 1—This code is part of the applet's definition of notifyChange, which is called whenever a variable is changed at the interface, showing the interactivity between the device and the interface.

```
public void notifyChange(String variable, Object value){
    if(variable.equals("Sec")){
        seconds = ((Integer)value).intValue();
        timeDisplay.setSeconds(seconds);
    }
    else if(variable.equals("Hrs")){
        hours = ((Integer)value).intValue();
        set_hours();
    }
    else if(variable.equals("Min")){
        minutes = ((Integer)value).intValue();
        timeDisplay.setMinutes(minutes);
    }
}
```

device is intelligent enough to turn back on to a preset schedule after an allotted time without receiving any data input.

PUTTING THE PIECES TOGETHER

Everything sounds good in theory. But what about putting this smart device in place?

In addition to about \$20 worth of garden-variety sprinkler components, I used an AT89C2051 microcontroller. This 20-pin 8051 derivative from Atmel includes 2 KB of program space.

I installed an RS-232 transceiver to provide external communications. The software handles a real-time clock/calendar and an alarm clock that sequences throughout the program and manages the watering start times.

The emMicro module provides a remote interface for manually programming the system and customizing watering schedules. One of the beauties of the remote interface is that it can be easier to use than the physical

interface on the device itself. Some sprinkler controls seem to require a college course just to get your lawn watered at the right time.

A serial EEPROM device stores interface documents and program times in a nonvolatile array to ensure that the device retains essential information if it loses power. A 9-V battery serves as a power backup to keep the clock running during a power failure.

All you need onboard is the microcontroller and the hardware it needs to turn on 24-V AC devices. I used a 74HC138 decoder so eight valves could be controlled using three lines from the microcontroller. The schematic provides for an additional decoder that gives control for up to 16 valves (see Figure 2).

Figure 3 shows the programming logic. As I mentioned, the real-time clock/calendar was implemented completely in software, and it uses the microcontroller's 18,432-MHz crystal as its time base.

The clock has two main software components—an interrupt-service routine (ISR) and the clock routine. The 8051's hardware timer 0 executes the ISR once every 10 ms. The ISR must reset the hardware timer for the next interrupt and account for interrupt latency, which would cause the clock to drift off.

Unfortunately, the 8051 architecture doesn't support a 16-bit auto-reload mode, which would make it unnecessary to correct for interrupt latency. This situation isn't hard to correct.

Because T0LOW (an 8051 special function) is small (zero), there's no real possibility of a rollover when the addition is done and, therefore, no reason to account for a carry into the upper portion of the counter. The most important thing the ISR does is set the bit variable called `temMSbit`, which tells the real-time clock routine that a time tick occurred.

The clock routine is responsible for counting the ticks created by the ISR. Basically, it just counts hours, minutes, seconds, and days, and it sets a few bits here and there to tell other routines when to operate.

For instance, once every minute, the bit variable `CheckWater` is set. This way, the alarm-clock routine knows to check all the scheduled watering times to see if any are supposed to begin.

There are two day counters. One is used for watering schedules based on the day of the week (Monday through Friday) and the other handles periodic watering (every other day, every third day, etc.).

The counting routine is called by the main calling loop in a round-robin multitasking methodology. It is called much more frequently than the 10-ms rate of the timer ISR, so it is sure not to miss a tick.

Whenever the ISR sets `temMSbit`, the clock routine increments its clock registers as necessary. It then clears `temMSbit` so it won't respond more than once to each tick.

PROGRAMMING BRAINS

Three main steps were involved in integrating EMIT software into the sprinkler-system application:

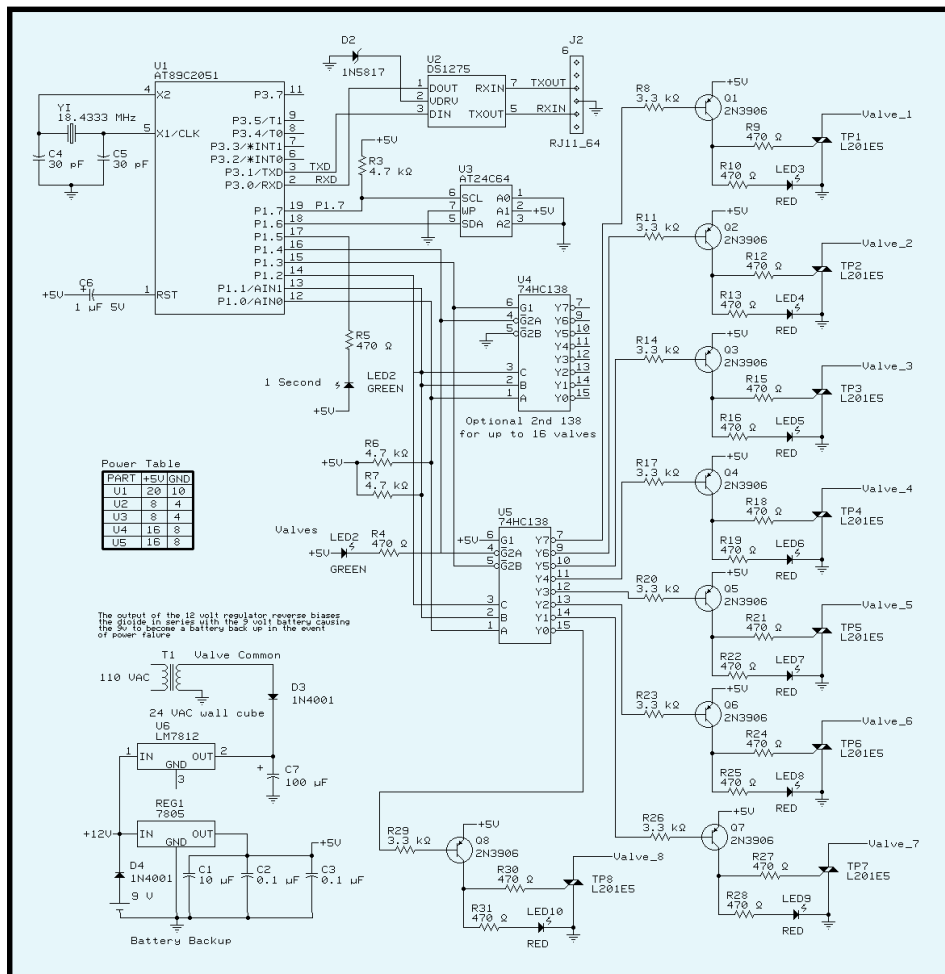


Figure 2—The connections made in the emWare sprinkler controller enable external information to control the function of sprinkler valves.

- include the emMicro code module in the application
- create the resource tables
- insert calls into the entry points of emMicro

The first task is usually handled via an `include` statement placed after the application code.

Information about the application is specified for the browser interface from the resource tables. Creating the resource tables is just a matter of editing the example tables included with EMIT to match the application's needs.

For example, my sprinkler controller primarily uses the variable table to make a large number of variables (e.g., Hrs, Min, and Sec) visible to the web-browser interface. These tables tie the browser interface to the microcontroller application. Listing 1 presents a section of the variable table.

The structure of each table is similar. The first byte is the number of elements in the table. This sprinkler controller uses 37 variables, so the first byte in the variable table is 37.

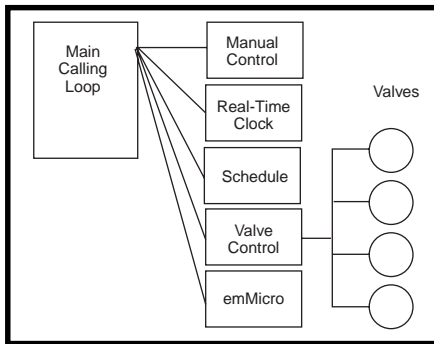


Figure 3—The programming logic centers around the main calling loop, which receives information from and transmits information to other controlling elements of the device.

Next, EMIT components process user requests to view and set device information. There are two bytes to describe the type of each variable (byte or word) and whether the variable is read-only or read/write.

Finally, variable names are placed into the table in the form of NULL-terminated strings. Other tables specify events, functions, static documents, and capabilities. Once these tables are created, they are included at the end of the application source code either via

include statements or by directly including the tables.

Placing calls to the emMicro entry points is straightforward because there are only two. However, there are some rules that applications must obey.

One entry point is the serial port interrupt vector entry. emMicro has its own serial ISR, but the vector has to be installed into the microcontroller's main vector table. The other entry is the main emMicro entry (called emMicroEntry).

For an application to coexist with emMicro, the application must be written using a cooperative multi-tasking methodology. Each process, including emMicro, is called by a main calling loop, and it must respect all the others by releasing the process control back to the calling loop in a timely fashion.

"Timely," of course, depends on how often each process really needs to have the processor's attention. emMicro must be called at least often enough to receive each character from the serial port.

No process should ever stall the processor in a tight loop waiting for some event to happen. That's what interrupts are for.

The only other rule is that no process should get exclusive use of the CPU registers. Each process can use them without regard to their previous state.

This design has several advantages. First, the clock is set from the browser. Not a single line of code in the microcontroller application is occupied with how the clock is set. Also, all the configuration parameters are adjusted without any code in the microcontroller.

Forcing the browser to display new information (or fancy graphics) is as easy as changing the value of a variable in a resource table. If the microcontroller application calls for a special function to be executed when a button is pushed on the browser interface, you only need to write the code for the function. `emMicro` executes it for you.

BUILDING AN INTERFACE

Little code was needed to interface to the controller, since `EMIT` provides prebuilt `emObject` interface components and handles the communication details.

To simplify the task further, I used `Symantex's Visual Café` and integrated the `emObject` components with the existing component toolkit. This way, I could select the objects from the component palette, add them directly to the applet, and easily manipulate layout and component properties.

The heart of the applet is an invisible object called `emitjri` (`EMIT Java Runtime Interface`). A new `emitjri` object is configured to communicate with this controller, which has a manufacturer ID of four and a device ID of 11.

All `emObjects` are passed a reference to the `emitjri` object. This way, `emObject` events can directly set variables on the embedded controller. Changes in embedded variables can also change `emObject` properties directly.

`emitjri` also contains function calls that enable the applet to directly set and get embedded variable values, as well as registering a callback to be notified when an embedded variable value changes.

An example of an `emObject` directly controlling an embedded variable is

the watering on/off image switch. This object displays one of two images depending on its state (active or inactive).

You can establish the connection between the image switch `waterOnOff` and the embedded variable `Water`. So, whenever an `ActionEvent` occurs (when the switch is pressed and released on the interface), the event value (`True` or `False`) sets `Water`.

The embedded-controller clock is a good example of communicating variable state change to the display applet. The embedded variables `Hrs`, `Min`, and `Sec` change regularly, and these changes need to be relayed to the applet.

You may set up the callbacks for the embedded variables. Whenever the variable values change, `notifyChange` is called with the variable name and the new value. Similarly, we could

have omitted the `NotifyChange` callback for `Sec` by setting a direct link between `Sec` and the `emObject` `timeDisplay` (see Listing 2).

Since `emitjri` handled the communications requirements, the majority of the coding effort was spent defining event-handler functions. Whenever a button is pressed, text entered, list items selected, checkboxes checked, and so on, events are generated. The event-handling functions for 24-/12-h clock, A.M./P.M., and day select are characteristic of many event handling functions that had to be created.

Many features were built into the user interface (see Photo 1) that could not easily be accomplished via hardware or by defining the interface completely within the embedded controller. The size of the applet increased to approximately 75 KB, but the controller and

Listing 2—Events are generated every time an action is taken at the interface (e.g., text entered, boxes checked, items selected). This code describes the event-handling functions for the 24-/12-h clock, A.M./P.M. selection, and day selection.

```
// select 24-h clock mode
void hr24RadioButton_Action(Event event)
{
    // get value of embedded variable "State"
    Object object = emJri.getJSJriVariable("State");
    int state;
    // disable am/pm buttons
    amRadioButton.enable(false);
    pmRadioButton.enable(false);
    isClock12 = false;
    state = ((Integer)object).intValue();
    /* instead of using a separate variable (wasted resource) to
       hold clock mode, set bit 0 of the State variable to indicate
       24-h clock mode. */
    emJri.setJSJriVariable("State", new Integer(state | 0x0001));
    set_hours(); // update clock
}

// change clock to am
void amRadioButton_Action(Event event)
{
    /* reset embedded variable "Hrs." Device runs in 24-h mode so
       subtract 12 from current hour. Since there is a JriLink to
       this variable, notifyChange will be called and complete the
       clock update */
    emJri.setJSJriVariable("Hrs", new Integer(hours-12));
}

// set current day
void dayChoice_Action(Event event)
{
    int day = dayChoice.getSelectedIndex();
    /* set current day from 0 to 6 depending on the selected item */
    emJri.setJSJriVariable("Days", new Integer(day));
    dayLabel.setText(dayNames[day]);
}
}
```




Photo 1—The GUI can match the device interface or add to the functionality available on the device. This weather-station interface provides an easy, intuitive view of weather conditions, enabling the user to see the impact of changing data on the device functions.

support interface hardware costs were reduced.

The end result is a Java class file that we can compress onto a 32-KB EEPROM and serve from the device.

Or, emGateway could serve the client the prestaged class file along with the standard set of preinstalled emObjects.

EDUCATION IS POSSIBLE

When the PC came out, its intelligence was amazing. But the PC only truly became intelligent when it was networked with other devices and connected to the Internet.

Embedded devices are following suit. Their limited intelligence expands as they connect to the Internet and other networks.

This isn't just a vision for devices with 32- and 64-bit micros with their extended memory and costs. And, you don't have to settle for a stripped-down web server.

Tremendous potential still exists in the 8- and 16-bit microcontroller market. That growth will continue as devices using these smaller MCUs are connected through a complete device

networking platform, which includes access to the Internet.

Intelligent devices are a reality, so long as we are intelligent enough to connect those devices for continuing education. ☐

Chris Sontag is chief technology officer and a cofounder of emWare. His areas of expertise include networking, security, X.500-based directory services, embedded systems, and internationalization. You may reach Chris at csontag@emware.com.

SOURCES

EMIT V.2.5 Software Developer Kit

emWare Inc.
 (877) 4-emWare
 (801) 256-3883
 Fax: (801) 256-9267
www.emware.com

AT89C2051

Atmel Corp.
 (408) 441-0311
 Fax: (408) 436-4200
www.atmel.com

FEATURE ARTICLE

Beau Wadsworth

Smart Building-Control Applications

Dreaming of those smart buildings of the future? Well, it's time to wake up. Beau's powerful machine-learning software figures out building occupancy patterns, yet it's simple and small enough to run on any microcontroller.



If you're like me, you've been waiting a long time for the arrival of the smart building. In theory, our homes and offices will be outfitted with intelligent lighting and energy-management systems that learn our habits and adapt to our needs.

Many people have done research on advanced building-control software using the PC, or some other "large" platform, intent on building a truly intelligent system. However, there haven't been that many applications deployed in the marketplace.

In this article, I want to show you a software technique that exhibits machine learning but in a small, simple implementation that runs on any microcontroller. The algorithm tries to understand the comings and goings of people and to anticipate when they might arrive next so it can provide advance heating, cooling, lighting, and so forth.

NOT-SO-NEW TECHNOLOGY

This type of intelligent building software was first used, I believe, in certain HVAC controllers as far back as 1987. Used mostly in the lodging industry, these controllers are

designed to squeeze every ounce of energy savings while providing adequate comfort.

In my application, the software is divided into two major components. The Occupancy Detection Unit (ODU) must recognize not only when the area is actively occupied and when it is not occupied, but also when the area is occupied by inactive occupants, who may be resting or sleeping.

The Pattern Learning Unit (PLU) records the occupancy history for the area and predicts the next arrival.

OCCUPANCY DETECTION UNIT

The ODU provides the information that the PLU uses to predict future activity. It may seem like a no-brainer to detect occupancy in software (assuming hardware input like a motion detector), but the task is tougher than it looks. In fact, it's difficult to know for sure that no one is in the area.

The problem is this: when motion activity ceases for a period of time, how do you know what happened? Are the occupants gone? Is someone there who is simply inactive? The solution is to detect two different types of activity—interior activity and entry/exit activity.

As shown in Figure 1, you can accomplish this task by monitoring the door(s) of an area (via door switches used in the security industry) and also monitoring the interior of the area via motion detection. Any time an entry/exit door is opened, this signals a potential change of occupancy.

As Figure 2 illustrates, it's not hard to figure out whether people are coming or going. If there is no motion detection following activity at the door, then someone must have left.

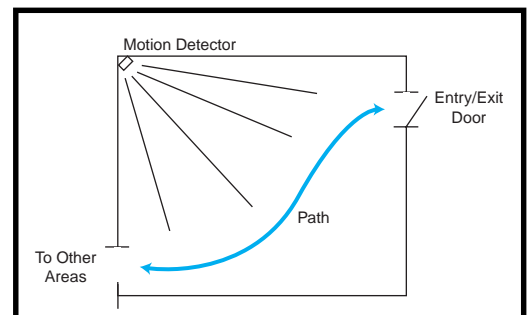


Figure 1—A combination of entry/exit-door and interior-motion information gives the needed results.

Conversely, if motion detection simply ceases without any door activity, then the area is not really unoccupied. Instead, the occupants must have retired to an inactive state even though they are still in the building or area.

The accuracy of this method is tied directly to the quality of motion-detector coverage—the more the better. In a house or small office, simply covering the main traffic area gives good results. Using motion detection in every room gives near-perfect performance.

In the algorithm, the search process starts when the entry/exit door is used. After the door closes, we look for motion for a period of time. If there is activity, then clearly someone is still on the premises. If not, then you may declare the area unoccupied.

The time period used can vary, but a good figure is in the 10-min. range. The delay period is a trade-off. Too

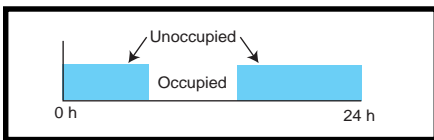


Figure 3—Here's a typical occupancy history record for a 24-h period. This system doesn't even know what the actual time is. It simply starts recording at powerup and rolls over 24 h later. Each day-long record is matched against previous records.

little time and you find that the area is frequently declared unoccupied, only to have this decision reversed a short time later when an occupant emerges from some back room. Too much delay time, and energy savings are reduced while the algorithm waits for occupants who may never appear.

One issue affecting the ODU is the phenomenon of slow motion-detector release. The occupancy-detection algorithm looks for motion detection as soon as the door closes. However, some motion detectors stay tripped for a relatively long time after motion ceases (up to 6 or 8 s), which causes a problem when the slow motion detector is located near the entry/exit door.

While leaving, the occupant trips the detector, opens the door, steps out, and closes the door. Unfortunately, the motion detector is still triggered even though the occupant has stepped out and closed the door! You can see

this in Figure 2a as the overshoot relative to the door timing.

This problem forced me to add a blanking window to the algorithm so that the real search for motion is delayed by a few seconds. A value of 6 s works well. Any more than that and the algorithm often fails to detect when someone arrives and passes through the motion-sensing area too quickly.

In that case, the system sees the arrival but turns around and looks for evidence of someone leaving (remember, door operation starts the test process). When the system sees no motion activity, except what took place during the blanking period, it assumes the person left again. So, the blanking window has to be set carefully.

As well, a false unoccupied state can be declared when someone leaves the area while another person, who has been inactive all along, is hidden from motion detection. Clearly, because of this weakness, the technique should be used with great care.

Many types of automatic responses (e.g., arming a security system) should not be implemented. While the system's performance can be increased by adding more motion detection, there's always the possibility (especially where occupants may be sleeping) of a misfire.

PATTERN LEARNING UNIT

Once the ODU has a good idea of the occupancy state, you can put this information to use. The PLU makes a continuous record of the occupancy state using a simple encoding means.

Three bytes contain a total of 24 bits, each of which can represent the occupied or unoccupied status for any one hour in a day. Under this method, any one-day record can be encoded in three bytes. An entire week's worth of data takes only 21 bytes. Later, I'll show you how this encoding method makes it easy to compare occupancy patterns.

As I mentioned, in this implementation, the system doesn't even know what the actual time is. It simply starts recording time from powerup and notes the passing of hour, day, and week increments. As shown in Figure 3, a record for any given "day"

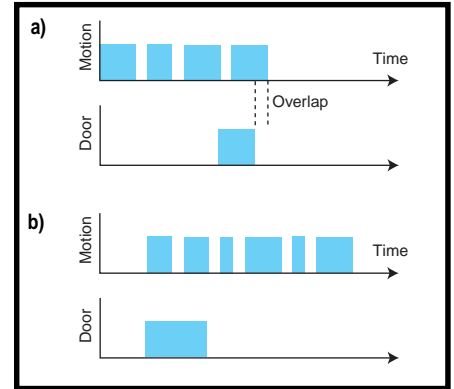


Figure 2—From the timing relationships, it's easy to see when occupants leave or arrive. **a**—In the leaving situation, an overlap can occur which must be accounted for in software. **b**—This graph shows an arrival.

indicates clearly what the occupancy pattern was for that day.

THE MOMENT OF TRUTH

The core technique at work in the PLU is to take the record for the same day of the previous week and use it to predict the occupancy for the current day. It's well worth keeping a week's worth of records since human habits center around a weekly pattern.

The process of putting the data to work is straightforward. Once every hour, at the same moment when the current occupancy state is recorded in today's record, the record for that same day in the previous week is evaluated.

At this moment of truth, the algorithm looks ahead in the record to see if there is any expected occupancy. It only needs to see a couple of bits in the data from the previous week—the ones that indicated the occupancy state in the next two hour slots.

These bits tell us if the area was occupied a week earlier during the upcoming couple hours. Whenever the

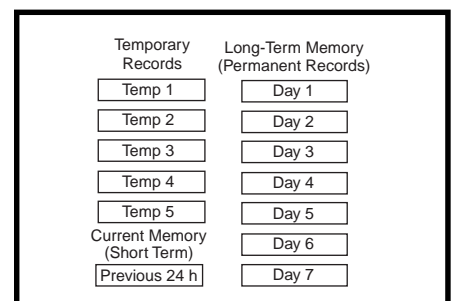


Figure 4—The system keeps track of previous patterns. Short Term Memory is the most recent 24-h record. Long Term Memory is a week-long record that remains semipermanent. Five temporary slots hold unusual records.

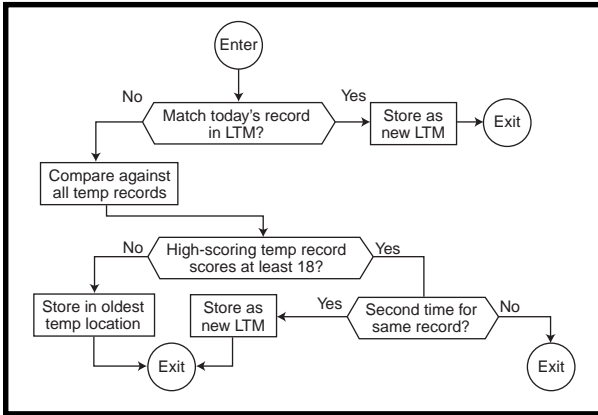


Figure 5—This block diagram shows the overall strategy. The current day's record is compared to various patterns in an attempt to find the best match. It looks first for a close match with the most likely element—that day's record in LTM. Failing that, it looks next for a sufficiently high score with any of the temporary records.

building is unoccupied, it's simply a matter of testing the next two hour bits, and voilà—we have anticipation.

Using this method, with 1-h resolution, accuracy is ± 1 h. That's why the anticipator looks at the next 2 h of last week's record. If it only looks 1 h ahead, it could err on the minus side all the way down to (almost) zero anticipation. Using 2 h gives it a performance range of 1–3 h of anticipation.

The range can easily be decreased by using a higher resolution system-wide. For example, records can indicate half-hour increments instead of an hour. This technique uses double the amount of RAM but has an anticipation range from 30 to 90 min. The beauty of this method is that any resolution can be used and the implementation stays pretty much the same.

A FUZZY RECOGNIZER

It may seem like the PLU was relatively easy to construct. Unfortunately, the algorithm is still not very smart.

Even though it predicts the arrival of occupants with some success, it records and acts on any and all patterns that arise, regardless of how atypical they may be. To build a sturdy machine that only recognizes typical patterns, a more complex mechanism is required.

The software approach that transcends the basic PLU is a sophisticated fuzzy recognizer that verifies the ordinariness of a pattern before committing it to memory. Note that I'm using the term "fuzzy" loosely. It does not refer to the use of fuzzy logic

as an engineering approach. Here, "fuzzy" means that the software is able to recognize patterns even when patterns don't match exactly.

The algorithm also recognizes when the building is unoccupied for an extended period (vacation detection) and suspends all pattern recording and anticipation until the building is normally occupied again. The original patterns are retained until they are needed again.

The fuzzy recognizer makes a series of comparisons that each day's record is put through in an attempt to gauge its validity. Figure 4 shows how previous records are stored in memory.

Long-term memory (LTM) is a semi-permanent record, one week long. If a current record passes muster, it replaces the existing record for that particular weekday in LTM.

There are five additional slots for temporary records that did not match anything at the time they appeared.

The temporary records are key in the process of learning new patterns. Anytime a current pattern matches the same temporary pattern for two weeks running, the system assumes that it's seeing a new schedule for that weekday.

Figure 5 shows a block diagram of the matching process. Each day's record is compared to the same day's record in LTM using a pattern-matching process. As shown in Figure 6, the two patterns are compared bit by bit with the total number of like bits being noted.

Clearly, if all 24 bits are the same, the patterns are exactly alike. The tricky part is recognizing patterns that are very similar but not exactly alike.

The approach used here was to shift the patterns one bit position relative to each other and then measure the score again. This technique tells us if the patterns are similar except for being offset somewhat.

As you see in Figures 6a and 6b, the shifting process can yield higher

scores for patterns that really are similar, while producing nothing for patterns that aren't all that much alike. Shifting both to the left and to the right and then looking for a score of at least 23 out of 24 gives a good assurance of a similar pattern.

If the record is not deemed similar enough to the LTM record for that weekday, it is compared to the five temporary records. The comparison with the temporary records is similar to that for LTM records, except that the shifting process is not employed and the current record is compared to all five temporary records with the score of each comparison being noted.

The highest-scoring comparison that scores at least 18 is taken as a match. A current record that fails to match any record is deemed a new temporary record, overwriting the oldest one in memory.

If a given record does match a temporary record, a quick check is done to see if the same thing happened last week. If the record matches the same miscellaneous record for two weeks running, that record wins a place as the LTM for that weekday.

I should note that the algorithm also checks for a completely unoccupied state during the current record's 24-h period. Whenever there is a match with the unoccupied profile, a counter is incremented. If this happens for three consecutive days, all system learning is suspended (i.e., vacation detection).

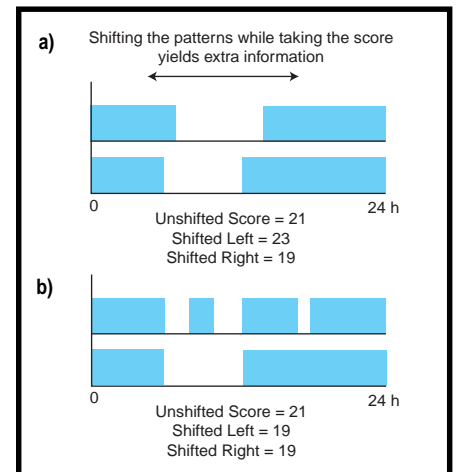


Figure 6—When patterns are compared, each bit position is checked and the number of matching bits are counted to yield a score. Shifting the patterns one bit to the left and to the right tells us if the patterns are similar but simply offset by an hour or so.

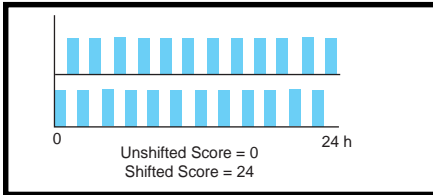


Figure 7—The comb problem can yield “matches” between records that appear similar but actually are almost opposite.

Through this pattern-matching process, a delicate balance is created. If a pattern is similar to patterns for that weekday in the past, the new pattern is immediately adopted as permanent. This technique enables quick adjustment to changes like Daylight Saving Time.

If the pattern isn’t similar enough to adopt immediately, it is compared to previous unusual patterns. If there is a match with one of these for at least two weeks running, then that pattern is taken as the permanent record.

As a result, slight pattern changes are adopted immediately, while anomalous patterns are ignored unless they persist for several weeks. In that case,

they are adopted as the new permanent record.

BUT DOES IT WORK?

The system performed very well during informal testing. As expected, the algorithm learns occupancy patterns within a maximum of three weeks.

When schedule changes are slight, the system doesn’t change for the following week. When given a markedly different pattern, the software waits a couple weeks before adopting the new pattern into its permanent record.

Because the algorithm is verifying the pattern’s validity in the second week, it’s too late to act on it even if it passes the test. In the third week, the anticipator responds to a new pattern.

This algorithm has one weak spot. When the patterns contain a large number of alternately occupied and unoccupied periods, as Figure 7 shows, the algorithm is threatened by what I call the “comb problem.”

The problem is that the shifting process used in the matching algorithm can yield a high score for patterns that

look similar (like two combs) but that are almost opposite in timing.

Needless to say, this mode of operation is unlikely because of the rarity of a comb-like pattern arising in normal circumstances. And even if the comb problem appeared, it may be irrelevant because the anticipator looks ahead for some time.

In other words, if the area saw very frequent occupancy changes, the anticipator ends up expecting arrival pretty much all the time. Fortunately, this is the behavior you want anyway, so the comb problem is fairly benign.

SYSTEM RESOURCES

In case you want to set up a similar system, you can download the source code for the PLU. However, I didn’t include code for the ODU. It is intertwined with the application software and would have been difficult to use in that form.

The PLU takes only a few hundred lines of assembler code, which translates to about 750 bytes of storage on an 8-bit micro. In terms of execution time, the worst-case condition takes about 1800 instructions.

On an 8031 running at 12 MHz, that equates to about 5 ms to run the PLU thread. This light load is further eased by the fact that the bulk of the PLU thread only runs once per day for an execution overhead of roughly 0.0000057%. So, the PLU can be easily integrated into an application without burdening system resources.

I hope you’ve seen how straightforward it is to set up intelligent building-control applications. Now you’re ready to take us into those smart buildings we’ve heard about for so long. 🏠

Beau Wadsworth has been designing embedded systems since 1983. His company designs and manufactures products on an OEM basis for the security and home-automation industry. You may reach Beau at (423) 689-8851 or at b_wadsworth@nxs.net.

SOFTWARE

The source code for the PLU is available on the Circuit Cellar web site.

Photo courtesy of emWare

EMBEDDED

NOVEMBER 1998

- 40** Nouveau PC
edited by Harv Weiner
- 44** All BIOSs are Not Created Equal
Scott Lehrbaum
- 49** Real-Time PC
Embedded RT-Linux
Part 1: General Introduction
Ingo Cyliax
- 57** Applied PCs
emWare Top to Bottom
Part 1: Monitoring via the Internet
Fred Eady



SUPER-FAST CPU BOARD

The VME64 (V2P2) and CompactPCI (C2P2) are believed to be the industry's fastest SBCs. Both feature a pair of Pentium II Deschutes processors operating at speeds up to 450 MHz (550 MHz in the future). Each processor is equipped with up to 512 KB of level-2 cache to maximize memory-access performance.

The two processors share up to 1 GB of 100-MHz synchronous DRAM main memory. The processors, cache, and memory are linked via a 100-MHz FSB Local bus, which supports the 450-MHz Deschutes processors as well as next-generation 550-MHz Katmai processors. The boards incorporate Intel's 82443BX and the 640 Advanced Graphics Processor. The DEC 21554 Draw Bridge Chip is included on the CompactPCI version to enable multiprocessing.

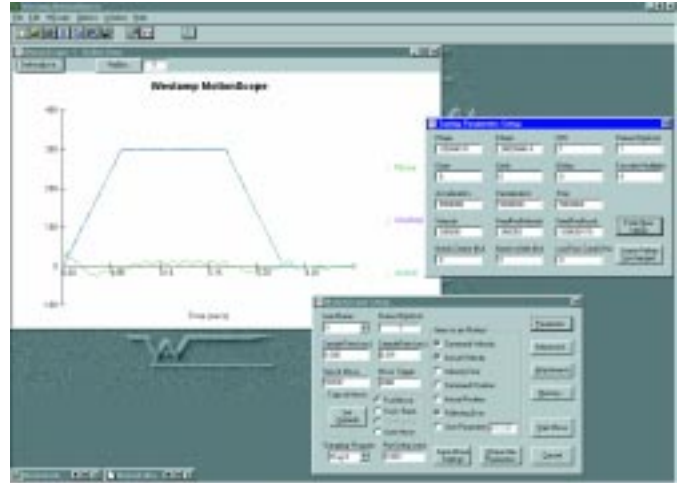
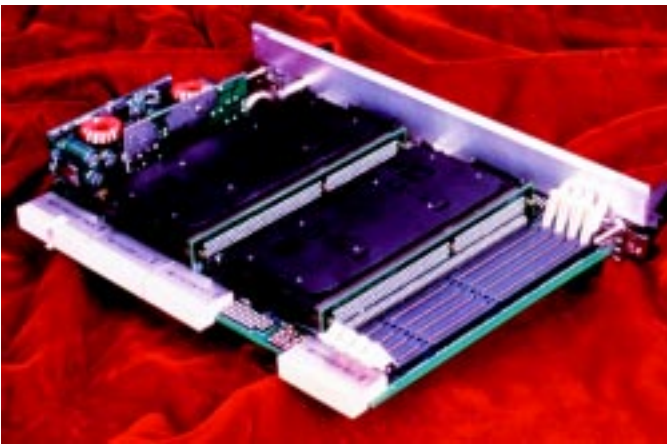
The boards also include dual Ethernet interfaces (twisted pair) operating at 10 or 100 Mbps, a 40-Mbps ultra-wide SCSI interface with auto-termination, and a 64-bit AGP graphics engine with 8 MB of video RAM optimized for 3D rendering. Also available are two ultra-DMA 33 IDE interfaces, a pair of USB ports, and a parallel port.

For applications that must be deployed without a rotating hard disk, the boards provide 72 MB of M-Systems' DiskOnChip flash memory and up to a 750-MB SanDisk 1.5" flash IDE. The DiskOnChip flash memory comes with M-Systems' TrueFFS file-system software, which enables the flash memory to emulate a hard disk.

The V2P2 and C2P2 run a variety of OSs, including Win NT, Solaris x86, QNX, Vx-Works, and Real I/X. The boards come equipped with AMI's BIOS and onboard diagnostics software and status LEDs.

Pricing for the V2P2 and C2P2 starts at **\$7200** with 0.5-GB memory and two 400-MHz Pentium II processors.

General Micro Systems, Inc.
(800) 307-4863 • (909) 980-4863
Fax: (909) 987-4863
www.gms4vme.com



MOTION-CONTROL DEVELOPMENT SYSTEM

MotionObjects is an object-oriented application that turns a Windows 95 or Windows NT computer into a digital oscilloscope and motion-control development system. It complements Westamp's SP2k-series multiaxis digital-positioning brushless servo driver. Three main development tools help program and fine-tune the SP2k servo drivers—MotionEditor, MotionLink, and MotionScope.

MotionEditor is a program editor that enables the user to drop in text and edit it into a usable program. Working in conjunction with MotionEditor, MotionObjects offers a compiler that enables the user to code in a highly intuitive, procedural, and object-oriented language.

MotionLink lets the user speak directly to the hardware from any Windows 95- or NT-based PC. Using an RS-232 cable, the user can download or upload a program into the SP2k or similar servo drivers.

MotionScope is a feature-rich graphics-plotting tool used to evaluate and optimize system performance. The data files may be saved for later retrieval, plotting, printing, and analysis. MotionScope plots command speed (programmed speed), actual speed, and following error. Tuning variables include integral gain, derivative gain, velocity feed-forward gain, and acceleration feed-forward gain enabling the programmer to fine-tune the SP2k servo driver for optimal performance.

MotionObjects sells for **\$395** with the purchase of the SP2k-series digital-positioning brushless servo driver.

Westamp, Inc.
(818) 709-5000
Fax: (818) 709-8395
www.westamp.com

Nouveau PC

edited by Harv Weiner

CompactPCI SBC

The **SC-1501** is a CompactPCI single-board computer that supports Intel Pentium CPUs ranging from 133- to 200-MHz MMX. The single 6U-

slot-size unit features a built-in 64-bit GUI accelerator with 2 MB of DRAM, up to 128 MB of synchronous DRAM, 10BaseT and 100BaseTx Ethernet, and USB port. It also includes 512 KB of cache and a 128-KB flash BIOS.

Other features include a floppy-drive controller, one ECP/EPP parallel port, two 16C550-compatible serial ports, real-time clock, flash-disk option, and support for the 32-bit CompactPCI bus and DEC PCI/PCI bridge. An extension file module card—the SC7001—adds support for a 2.5" hard disk drive, 3.5" floppy disk drive, and optional SCSI and PMC slot interfaces.

Computer Modules, Inc.
(408) 496-1881
Fax: (408) 496-1886
www.compumodules.com



INDUSTRIAL SBC

The **VSBC-6** is an EBX-compliant single-board computer that includes PCI-based video, flat-panel BIOS support, 10BaseT Ethernet, PC/104-Plus expansion (the PCI-enhanced version of PC/104), and USB interface. The new embedded computer is compatible with QNX, Windows 95/98/CE/NT, VxWorks, and other popular operating systems.

The 8" x 5.75" x 2" board includes a full complement of industrial features such as eight 12-bit analog channels, 16 opto-22 digital lines, two RS-422/-485 COM ports (plus two standard RS-232 COM ports), and three extra timer/counters. For high-reliability applications, the board includes ECC circuitry that detects and corrects one-bit memory errors on the fly, as well as a watchdog timer with true hardware-reset capability.

The VSBC-6 accepts all socket-7 processors up to 300-MHz 'K6 and 233-MHz Pentium. Memory options include 8-128-MB EDO or true error-correcting SDRAM, 2-72-MB DiskOnChip flash memory, 512 KB of battery-backed SRAM, and 256-KB level-2 cache.

The VSBC-6 board includes a highly reliable design and construction, with latching I/O connectors and latching high-reliability memory sockets. Each board is subjected to a 48-h burn-in and 100% function testing, and is backed by a two-year warranty.

The VSBC-6 is priced at **\$793** with a 200-MHz 'K6 CPU in 100-unit quantities.

VersaLogic Corp.
(800) 824-3163
(541) 485-8575
Fax: (541) 485-5712
www.versalogic.com



Nouveau PC

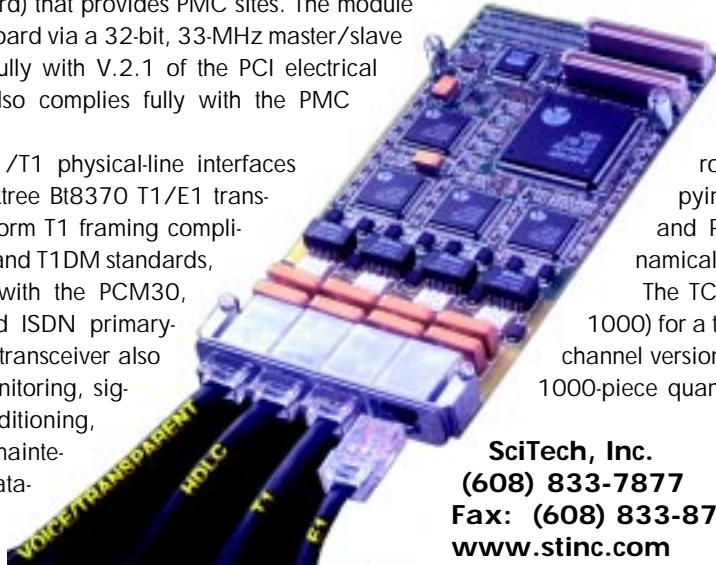
PCI MEZZANINE CARD

The **TCOM4-SC** is a PCI mezzanine card that provides four E1, T1, or primary-rate ISDN interfaces.

Both long- and short-haul communications at speeds of 1.544 Mbps (T1) or 2.048 Mbps (E1 or primary-rate ISDN) are supported.

The TCOM4-SC can be used with any baseboard (e.g., a VME bus or CompactPCI CPU board) that provides PMC sites. The module communicates with the baseboard via a 32-bit, 33-MHz master/slave PCI interface that complies fully with V.2.1 of the PCI electrical specification. The module also complies fully with the PMC mechanical specification.

The TCOM4-SC's four E1/T1 physical-line interfaces are implemented using Brooktree Bt8370 T1/E1 transceivers. The transceivers perform T1 framing compliant with the SF, ESF SLCR99, and T1DM standards, and E1 framing compliant with the PCM30, G.704, G.706, G.732, and ISDN primary-rate ISDN E1 standards. The transceiver also supports alarm and error monitoring, signaling supervision, trunk conditioning, and facility-data-link (FDL) maintenance. The TCOM4-SC's data-link interface is based on Brooktree's Bt8474 HDLC



controller, which can format and deformat up to 128 HDLC channels.

To support low-latency communications with other boards residing on a common backplane, the TCOM4-SC provides an optional SC-Bus (Signal Computing Bus) expansion port based on VLSI Technology's SC4000 controller. Optimized for carrying multimedia traffic such as voice, fax, and video, the SC-Bus enables up to 128 channels of incoming or outgoing T1, E1, or PRI traffic to be routed to other boards without occupying system-bus bandwidth. T1, E1, and PRI channels can be mapped dynamically to any of the SC-Bus time slots.

The TCOM4-SC starts at **\$465** (quantity 1000) for a two-channel E1/T1 solution. A four-channel version of the TCOM4-SC costs **\$635** in 1000-piece quantities.

SciTech, Inc.
(608) 833-7877
Fax: (608) 833-8738
www.stinc.com

CompactPCI DEVELOPMENT SYSTEM

The Ziatech **ZT 5081** 6U 14-slot development system features an SBC with a 200-MHz Pentium/MMX processor, 64-MB ECC DRAM, 512 KB of L2 cache, a 3.2-GB IDE hard drive, a floppy drive, a wide SCSI interface, and two 150-W hot-swap power supplies. The feature set is housed in a 19" rack-mount IEEE 1101.10-compliant Compact-PCI chassis with a 14-slot backplane, suitable for general-purpose and telecommunications applications.

In addition to accommodating 32- or 64-bit Compact-PCI cards, the backplane supports the H.110 computer telephony bus for voice and other telephony applications and IEEE 1101.11-style rear



transition cards for simplified cabling. The backplane also supports peripheral and I/O cards adhering to the PICMG hot-swap.

The new rack-mount system comes with MS-DOS, Ziatech's industrial BIOS, and flash disk. The internal hard-disk drive is ready for the installation of PC-compatible operating systems such as Windows NT, QNX, or VxWorks. Ziatech offers optional development toolkits complete with additional files to help simplify the implementation of these operating systems.

The ZT 5081 6U 14-slot development system is priced at **\$5595**.

Ziatech Corp.
(805) 541-0488
Fax: (805) 541-5088
www.ziatech.com

Nouveau PC

All BIOSs are Not Created Equal

No PC-based CPU is suitable for embedded applications without an equally tough BIOS. Scott helps us make sure our embedded-PC products are fit enough to meet the exacting demands of medical, military, and factory environments.

Not long ago, anyone who suggested putting the PC architecture into an embedded system would have been laughed out of Silicon Valley. After all, the PC was designed to sit in a warm office playing solitaire, not venture out into the hostile climate of the embedded world. It was accident prone, too, and reliability is an absolute necessity in embedded systems.

But its limitations haven't stopped the PC. Among the PC's forays into embedded territory are PC/104, passive-backplane PCs, and lots of proprietary SBC form factors.

The PC architecture has become so pervasive, it's virtually a commodity technology. Almost anyone with a soldering iron and a cookie cutter can whip up a new embedded-PC form factor. But, can the resulting product be trusted to run a factory production line or test blood for bacterial contamination?

No doubt, every board vendor and form-factor proponent can argue their case vigorously. Some might hold up well,

especially architectures like PC/104 that were designed for embedded systems.

But, let's set aside the hardware and talk about BIOS instead. As much as board vendors might improve the average PC motherboard, no PC-based CPU is suitable for embedded applications without an equally fine-tuned BIOS.

PC SELF-DESTRUCT

To understand why "normal" PCs don't cut it for embedded systems, you don't have to look beyond your own desktop. Who hasn't seen their system crash?

It's not my intention to indict the PC architecture. The reasons behind some of these problems have nothing to do with how well the PC was designed.

But, it's important to remember that the IBM design team probably never dreamed that their sleek new computer would end up in supersonic aircraft and deep-sea submersibles. The issues they would have had to consider for such exotic applications never crossed their minds.

Take the CMOS battery, for example. For the average PC, a five- or ten-year battery life is probably more than sufficient (especially since a PC is considered obsolete after a couple years anyway). Not so with embedded PCs, which tend to have much longer operating lifetimes.

And what if the CMOS battery goes dead? With any other gadget, you just pop in a new battery. But on a PC, there's the small matter of system configuration data. If you lose it, you can be in big trouble.

What if this same basic technology was monitoring a patient's vital statistics during an operation? What if it was responsible for controlling a nuclear power plant?

Embedded applications take care of many critical functions. If they spontaneously combust, they cause a lot of damage.

LIKE KRYPTONITE

Obviously, a complete computer meltdown is never good. But not all of the PC's weak points pose such a threat. The effects can be more subtle, especially the ones

originating with the BIOS. But, they can still play havoc with embedded systems.

Before talking about ways of toughening up the PC BIOS, let's look at some of the special issues facing embedded systems in a few vertical markets (see Table 1).

Medical instruments jump out as an obvious example of systems that better work when they're needed. Not all medical devices are life-critical, but they may still have an impact on how a patient is diagnosed or what drugs are prescribed.

Medical systems are categorically intolerant of software crashes. No matter how well the application software is written, critical systems need a built-in automatic recovery mechanism. The machines also have to come up running very quickly.

Production-line systems can be mission-critical as well. Production lines churn out millions of dollars worth of product an hour, and down time quickly turns a profitable operation into a money loser. So, the embedded PCs have to be reliable, operating continuously for long periods of time.

Another common trait of automation systems is that the embedded PC tends to be deeply embedded. There's no floppy disk drive, keyboard, or mouse.

If there's a display, it's more likely to be a few black and white characters or color LEDs than a VGA monitor. For maintenance, some alternative method of accessing the system is needed.

There are all sorts of military applications for embedded PCs, from missile guidance to aircraft systems to ruggedized field PCs. Exacting specifications and operational reliability are an absolute must.

Many military applications are portable, which raises the issue of power management. To maximize battery life, a thorough and well-designed power-management interface is essential.

It's important to keep the system's total power appetite to a minimum, perhaps by using alternative, low-power options for program-storage and video-display features.

BIOS FITNESS

PCs usually work reliably, as long as you keep them in the shallow end of the pool. But there are some pretty effective ways of strengthening the PC, both at a hardware and BIOS level.

My program for getting an embedded-PC BIOS ready for mission-critical embedded applications is summarized in Table 2.

NO-FAIL STARTUP

Despite advanced PC technology, you can still run across ill-behaved hardware. Sometimes, systems quit for no particular reason during the power-on self-test (POST).

Whatever the problem, an embedded-PC BIOS can't just give up. It has to deal with recalcitrant hardware and coax it back into operation.

The BIOS has to keep retrying all available boot devices if the boot fails the first time through. The best thing it can do is restart POST in case any of the hardware wasn't initialized properly on powerup. This also gives the hard drive more time to get ready.

Many embedded systems don't have a keyboard or any other input device, so if the BIOS doesn't detect a keyboard, it can't report "Keyboard error or no keyboard present—press F1 to continue." It should continue POST no matter how the keyboard test comes out.

These are just two examples, but a well-designed embedded-PC BIOS needs an entire toolkit of problem-recovery options. That includes ensuring the hardware comes up running OK and that noncritical errors don't stop the boot-up process.

INSTANT ON

Ever timed your desktop PC to find out how long it takes to start up? Embedded systems can't get away with such outrageous startup delays.

Embedded-PC BIOSs should provide options for dramatically accelerating the start-up process. Aside from using a fast-booting OS, expert embedded BIOS developers can usually find lots of fat to trim in the POST procedure.

A good starting point is to cut features that the systems don't need, like repetitive memory tests and plug-and-play support. Some initialization routines can be streamlined if the hardware has a fast response time or even stripped out if its chance of failure is low. Obviously, that kind of change carries some risk.

For a super-fast startup, the BIOS may need to cut execution time. In this case, many of the hardware tests and initialization routines simply have to be skipped.

But if the board manufacturer can fully document what isn't being done, system developers might still want the feature. They may not need to use the parts of the system that aren't being initialized, or they might be able to initialize the hardware from inside their own application code.

BATTERY-FREE OPERATION

I talked about problems that can arise if the CMOS battery dies. But protecting configuration data isn't the only issue. Some embedded applications can't use batteries, especially if they're near explosive environmental gases. Hey, if a battery sparks while you're near a pocket of methane, you *become* the embedded system.

An embedded-PC BIOS should provide an alternative way of storing configuration data. For example, in its enhanced embedded-PC BIOS, Ampro uses a small EEPROM to back up CMOS RAM data.

The BIOS keeps a copy of the current set-up configuration in EEPROM and restores the CMOS if battery backup isn't available. The only information lost at power-down is the real-time clock time and date.

	Medical	Aviation	Telecom	IndustrialControl	Transportation	Military
High reliability	X	X	X	X	X	X
Alternative storage media		X		X	X	X
Power management						X
Watchdog timer	X	X	X	X	X	X
Battery-free operation						X
No-fail startup	X	X				X
Instant on	X	X				X
Unattended operation			X	X		
Remote access features		X	X	X	X	X
Start-up customization	X				X	X
RTOS support	X	X	X	X		X

Table 1—These critical embedded-PC BIOS enhancements are commonly required in certain vertical market segments.

ON STARTUP

You may not want end users to know that the magic in your box is a plain old PC.

One answer is to throw a logo onscreen early in POST and leave it there until the system boots into the application software. Or perhaps you can turn off the POST message display completely.

Another typical scenario is that the system uses an exotic video interface that requires special initialization early in POST.

Some applications grab control of the system early on and never let go. If the software doesn't need an OS and all the application code can be stuffed into ROM space, that's a nice way to cut the cost, weight, and power budget.

All PC BIOSs support the IBM-standard 55AA ROM scan, which gives ROM code like network boot ROMs and solid-state disks the opportunity to install themselves near the end of POST. The 55AA hook is a good starting point for giving the BIOS a greater degree of start-up flexibility.

A good embedded-PC BIOS gives ROM code many opportunities throughout POST to hook the BIOS and perform a special task before returning control to the system.

ALTERNATE ACCESS

So, your embedded system has no display or keyboard. How do you access it?

Lots of embedded applications don't need a user interface for system operation. Consider a system that records aircraft system status during flight and saves it for later analysis. It doesn't make sense for an OEM to add a video controller, display, and keyboard, increasing costs for features needed only for occasional maintenance.

The BIOS should offer options for remote console access over a serial port or other interconnect. It should enable keyboard commands to be entered and display information to be received on a host system.

Ideally, this access is achieved via a portable computer connected via a null modem cable or from a remote location using a phone line or radio modem. Since two or more serial ports are present on most PCs, these remote features provide console access at virtually no cost.

Then again, available serial ports may be scarce; they get used up quickly. It's good to let one of the ports double as a serial console access port. But, with no easy way to change the system set-up configuration, the BIOS needs a way to switch into serial console mode automatically.

REMOTE MAINTENANCE

You're also going to need occasional system maintenance. The embedded PC should let service personnel perform these tasks remotely.

Some users may want to keep their program source code and compilers resident on the embedded system so software modifications can be made on the system but from a remote location. Those systems need remote console access and a way to transfer files back and forth from a host.

Another requirement to making remote on-system software development feasible is on-the-fly serial debugging support. The embedded-PC BIOS has to support automatic download of the target portion of the serial debugger, rather than forcing that code to be ROM-resident on the target.

STORAGE SUPPORT

Embedded systems have to endure harsh environmental conditions. Intensive shock and vibration, temperature extremes, and strong magnetic fields cause most hard and floppy drives to go into the electronic version of anaphylactic shock.

Embedded-system BIOSs support a variety of options for using solid-state disk (SSD) media in place of mechanical drives. Without moving parts, SSDs are more rugged and less susceptible to breakdown. Ideally, the BIOS support should include removable SSD options.

Beyond SSDs, many embedded systems may be better off storing program code in ROM. This technique not only saves the expense of OS licensing but can mean that the application software launches faster.

WATCHDOG TIMER

It's a simple but essential requirement for mission-critical embedded systems: they must continue to work under all conditions. That includes software crashes, power brownouts, and more. Consider an auto-pilot application for a large jet. How long can that system be down?

A reliable watchdog timer (WDT) is a necessity for all systems that manage critical activities. The WDT monitors the system independently of the rest of the circuit, so it's not affected by system glitches. If something happens that causes the system to crash, the WDT can initiate a system restart and restore it to operation quickly.

Although a WDT is a hardware feature, the embedded-PC BIOS should support it through software interrupt services. The routines must give application software enough flexibility to manage the WDT and provide recovery procedures optimized to the application's requirements.

No-fail startup	Prevents hardware initialization errors and device failures from halting the startup
Instant on	Provides near-instant power-on to system-ready startup
Battery-free operation	Allows the system to operate without batteries, prevents a CMOS RAM backup battery failure from harming the system
Start-up customization	Allows customization of the startup process for system-unique hardware initialization, display of custom logos, etc.
Alternate console access	Provides alternative to standard keyboard and video screen via serial terminal or other interconnect
Remote maintenance capability	Allows remote software updates and remote debugging capabilities
Alternative storage media support	Provides options for program and data storage using nonmechanical (solid state) media
Watchdog timer	Restarts the system in case of a software crash, power brownout, or other failure condition
Power management	Reduces total system power budget for portable and battery-powered applications
Year-2000 compliance	Avoids software failures due to the Y2K computer bug
RTOS support	Optimizes system performance for compatibility with all major RTOSs
Unattended operation	Allows system to be operated for extended amounts of time with zero user intervention
High reliability	Fine-tunes system operation (signal timing, device initialization, etc.) for optimal survivability and performance under all operating conditions

Table 2—This list gives you a look at some of the benefits of embedded-PC BIOS enhancements to embedded applications.

POWER MANAGEMENT

Low power consumption is an important requirement of portable applications. PC BIOSs tend to have a lot of built-in support for automatic and software-managed power management. Initiatives like Green PC and power-management standards such as APM (Advanced Power Management) and ACPI (Advanced Configuration and Power Interface) have seen to that.

The changes to the power-management code are more subtle than other areas of enhancement. It's mostly a matter of optimizing power savings depending on the components and optional interfaces used.

If excessive temperature is an issue, the BIOS should monitor and control the CPU temperature. One approach is to automatically engage the CPU fan at a thermal setpoint. For power-sensitive applications, you can interleave CPU operation using a user-selectable duty cycle.

Y2K COMPLIANCE

Sure, this topic has been beaten to death recently, but it's important to see that

year-2000-related problems can affect embedded systems in much more dramatic ways than desktop PCs. Again, what they're doing tends to be more critical than the average desktop machine.

A Y2K-compliant BIOS should automatically update the real-time clock after a century rollover and should always calculate and return the proper century value.

RTOS SUPPORT

Support for real-time operating systems can be tricky. For one thing, they tend to discard BIOS interrupt services and access hardware directly.

Depending on how the BIOS has initialized the system components, different RTOSs may or may not like what they see. There's an art in refining the BIOS for any embedded PC to make sure the product is compatible with all RTOSs.

Another issue is how to access enhanced BIOS services from inside an application. RTOSs operate in protected mode, which is incompatible with virtually all BIOS interrupt routines. Special features that are supported through enhanced BIOS services may not be accessible to an applica-

tion program. A good embedded-PC BIOS provides some means for application software running under an RTOS to have full access to all extended system features.

AND THE REST

If extra storage space is available in a memory device onboard (e.g., the EEPROM used for configuration data storage), OEM customers usually find creative uses for it. For example, it may store a few bytes of application parameters. On the BIOS side, services are necessary to enable this OEM area to be read and programmed by application software.

A potentially useful option is a serial boot capability enabling the embedded PC to download boot code from a remote host. This setup can eliminate the need for any disk drives or alternative storage media on the target module.

Embedded systems may operate unattended for weeks, months, or even years. So, a well-designed embedded-PC BIOS eliminates any need for human interaction.

Since configuration data has been known to get lost from time to time, the BIOS should enable easy recovery from a bad or corrupt set-up configuration.

Finally, embedded PCs get used in all kinds of systems with different bus configurations and timing requirements. The BIOS should try to maximize the module's robustness by ensuring that component and bus timing parameters satisfy applicable specifications with room to spare.

A FEW LAST WORDS

To add the required enhancements effectively, an embedded-PC supplier needs experience serving the embedded market and knowledge about real-world applications. Embedded-system developers considering the PC architecture should check board suppliers to ensure that the BIOS is fine-tuned for embedded environments.

A well-designed BIOS is as important as ruggedized hardware. No matter how well the board has been hardened, it won't do any good if the BIOS buckles under pressure. [EPC](#)

Scott Lehrbaum has been at Ampro Computers for nearly ten years, where he has held numerous positions ranging from Software Engineer to Applications Engineering Manager. You may reach him at slehrbaum@ampro.com.

Real-Time PC

Ingo Cyliax

Embedded RT-Linux

Part 1: General Introduction

Ingo kicks off this miniseries on using Linux as an embedded operating system with an overview and a comparison to other freely available OSs. As we start seeing more and more projects that use Linux, it'll be pretty clear why they do.

In past columns dealing with real-time operating systems, I've mentioned Linux and explained how I use it as a development system and the primary OS on my laptop. Starting this month, I want to show you how to use Linux as an embedded OS—even as a real-time embedded OS.

I've been a Linux user and developer for about two years. I'm also a fan of other freely distributed Unix-like OSs, like NetBSD and FreeBSD. Both are direct descendants of the BSD (Berkeley Software Distribution) operating systems. I still use them for other projects. For example, my web and mail server runs FreeBSD.

But, Linux has such a huge following, I wanted to make sure you had the latest information. I also discuss some differences between Linux and FreeBSD/NetBSD.

INTRODUCING LINUX

Linux is an alternative OS for PC (and non-PC) platforms. There are really two aspects to what is generally considered

Linux—the kernel, which is the true Linux component, and the applications that have been developed and ported to Linux. The applications range from clones of simple Unix command-line utilities like `ls` and `vi` to C and Fortran compilers and even commercial applications.

Together, the Linux kernel and the extensive applications that come bundled with it are called distributions.

Also, the kernel and a number of applications are available under the GNU copyright, or GPL. Check out Pat Villani's series of excellent articles on FreeDOS (*INK* 95–96), where he describes the GPL and the issues concerning the embedded-systems integrator.

I should note that one of the differences between the Linux kernel and the NetBSD/FreeBSD kernel is that the Linux kernel is not GPL protected (whereas the others are) and it is subject to a different licensing agreement. Whether GPL is right for your project depends on your requirements.

Usually, it's not a problem. GPL only requires you to make available the sources for components that are already under GPL. Other components, applications, and modules that you develop can be excluded.

A real-time extender, which extends the kernel using a dynamically loadable module, is also freely available. It works similar to real-time extenders for OSs like Windows NT, by dividing applications into processes and threads that are real-time aware and standard user processes that go unaffected. There's more, but I'll get into Linux's real-time extension in a later column.

WHY LINUX?

Naturally, this question is tugging at your brain. There are several reasons to consider Linux as an embedded operating system for many applications.

For one thing, it's freely available and includes a TCP/IP stack. Also, it has multiarchitecture support for Intel, 68k,

<p>Ethernet Controllers</p> <p>3Com 3c501 (throw it away!) 3Com EtherLink II 3Com Etherlink Plus 3Com EtherLink16 3Com EtherLink III 3Com 3c590/3c595 Vortex Ansel Communications Model 3200 EISA Ethernet adapter Apricot 82596 ARCnet for IP driver Allied Telesis AT1700 DE425, DE434, DE435, DE450, and DE500 DEC EtherWORKS cards D-Link DE-600 Ethernet pocket adapter D-Link DE-620 Ethernet pocket adapter DEC DEPCA and EtherWORKS DE100, DE101, DE200, DE201, DE202, DE210, DE422 Digi RightSwitch SE-4, SE-6 Cabletron E2100 EtherExpress Pro/10 EtherExpress ICL EtherTeam 16i/32 EISA EtherWORKS 3: DE203, DE204, DE205 Fujitsu FMV-181/182/183/184 HP PCLAN/plus HP LAN HP10/100VG ANY LAN: J2577, J2573, 27248B, J2577, J2573, J2585 Shared-memory IBM Token Ring 16/4 AMD PCnet32, PCnetPCI NE1000, NE2000, and compatible NI5210 Ethernet NI6510 Ethernet Parallel Link Internet Protocol Sangoma S502/S508 series multi-protocol PC interface card SMC Ultra, SMC EtherEZ ISA SMC 9000 series Ethernet Starmode Radio IP DEC 21040, most 21*40 Ethernet AT&T GIS (nee NCR) WaveLAN Ethernet-like radio transceiver WD8003- and WD8013-compatible ether cards</p>	<p>SCSI Host Adapters</p> <p>SCSI driver for Symbios/NCR 53c700 series and 53c800 series host adapters BusLogic MultiMaster (NOT Flashpoint) SCSI host adapter driver NCR53c406a-based SCSI host adapter driver AdvanSys SCSI host adapter driver Adaptec AHA-152x host adapter driver Adaptec AHA-154x and 631x-based host adapter driver Adaptec AHA-174x host adapter driver Adaptec AHA-2740, 28xx, 29xx, 39xx, aic7xxx-based host adapter driver DTC 3180/3280 host adapter driver All DMA-capable DPT SCSI host adapters All PIO-capable DPT SCSI host adapters Future Domain TMC-16xx SCSI host adapters IN2000 SCSI host adapters Symbios/NCR 53C810, 53C815, 53C820, 53C825 SCSI host adapters Pro Audio Spectrum/Studio 16 IOMEGA PPA3/Parallel ZIP Qlogic FAS408 SCSI host adapter QLLogic ISP1020 SCSI host adapter Seagate ST-01/02, Future Domain TMC-8xx SCSI host adapter Trantor T128/T128F/T228 SCSI host adapter UltraStor 14F/34F (not 24F) SCSI host adapter UltraStor 14F/24F/34F SCSI host adapter WD7000-FASST2/WD7000-ASC/WD7000-AX/WD7000-EX SCSI host adapter</p> <p>CD-ROM Drivers</p> <p>Aztech CD268 CD-ROM driver Sony CDU-31A CD-ROM driver Philips/LMS cm20 CD-ROM driver GoldStar R420 CD-ROM driver ISP16/MAD16/Mozart soundcard-based CD-ROM driver Mitsumi CD-ROM driver Mitsumi XA/Multisession CD-ROM driver SoundBlaster Pro/Matsushita/Panasonic/Longshine/CreativeLabs/TEAC/ECS-AT CD-ROM Sanyo CD-ROM device driver Sony CDU-535 CD-ROM driver</p>
---	--

Table 1—Here is a list of some of the devices that are typically supported with a standard Linux kernel. In addition to these devices, many peripheral-card vendors also offer Linux-driver modules for their cards.

PowerPC, Alpha, Sparc, SMP, and others. So, you can obtain and begin using Linux with very little trouble.

One of the great benefits of using Linux is the large support network that exists for it, and let's not forget the fact that Linux supports many devices and bus architectures right out of the box. Of course, Linux is somewhat modular and there are many tools and programming languages available. With features like this, Linux starts to make more sense, huh?

Let's talk about these points in more detail. I already mentioned that Linux is freely available. Obviously, that's nice because you won't pay royalties for embedding Linux in your product. But, you need to make sure all of the modules and components you plan to use for your project are freely available. Most are.

You can download most Linux distributions from the Internet. But, this can be tedious over a slow link, so most of us opt to buy a CD-ROM. Yet, being able to download a distribution over the 'Net is extremely useful if you're in the field and need to upgrade you embedded system or get a new version of a device driver.

Linux has a TCP/IP stack built in. The TCP/IP implementation is very robust, since many Internet-service providers use Linux for their high-throughput web servers. The programming interface for the TCP/IP stack is the standard Socket API.

Linux runs on a multitude of architectures. I'm only going to talk about the Intel port, but it's good to know that you're not stuck with Intel. Some of the details in each port are different, but for the most part, Linux is Linux, at least at the application level.

Linux is ported to 68k, which includes VME-bus modules, Alpha, and PowerPCs. And, there's a port in progress for MIPS processors.

The US Robotics Palm Pilot may be the smallest system that Linux is ported for. It's a PDA based on Motorola's 68328 Dragonball processor. There's also work in progress to port Linux to smaller Intel processors, like the 80188.

Since Linux has such a great following, there's a large support network. Although many people provide Linux support by writing or porting applications and drivers in an effort to attain fame and glory, there are also several companies that are providing commercial support for Linux. Often, these companies reintegrate the fixes into the Linux kernel development effort.

There is yet another side effect to this large support group. Many of the fame-and-glory developers are students who have learned OS internals by observing the Linux kernel, and they'll eventually enter the job market.

Consequently, the number of software engineers familiar with Linux is growing, and they're even proficient at the kernel level. That's good news, and perhaps one compelling reason why much of Linux will stay freely available with the GPL.

There are a great number of device drivers available for Linux. Because Linux has been ported to so many different architectures, it supports various bus architectures, like ISA bus, PCMCIA, PCI, VME bus, S-bus, and others.

Many vendors of PC-compatible cards have Linux drivers available. While the quality of device drivers for Linux varies, many popular cards and chip sets are well supported. Also, there's hope of fixing or adapting the device driver for your purposes, since sources are available for most device drivers.

Drivers are available for just about all of the Ethernet chip sets, all standard AT peripherals (COM, IDE, floppy, etc.), and many VGA chip sets. Also, many CD-ROMs and SCSI devices are supported. Table 1 shows a list of devices that are supported in an older distribution of RedHat.

However, a flash file-system driver isn't supported. That's unfortunate. It would be nice for working with embedded systems. Hopefully, someone is working on it.

But, all is not lost—it's still possible to boot Linux from a flash file system. Flash disks, like SanDisk's CompactFlash devices, work under Linux because it acts just like a IDE drive. Next month, I'll look at these issues in detail when I embed Linux.

Linux supports dynamic module loading, which means libraries and devices drivers can be loaded by the OS after boot time to extend the base functionality of the kernel. Dynamic modules are nothing new, but they do make configuration easy.

You build a simple kernel that can be configured to deal with different device configurations by changing a configuration file. This construction also improves memory use, since only the required device drivers are loaded when needed. Of course, you can still configure a Linux kernel to be

static by compiling in all the device drivers it needs for a particular application.

Many applications and programming languages are available for Linux. This feature might not be important for building embedded systems, but it's nice to have access to all the applications on your development system.

The normal compiler used for Linux is the GNU C compiler (a complete C, C++, and Objective C compiler). The companion debugger (gdb) is flexible and can even deal with remote debugging via Ethernet or serial port. This capability means that you can debug your embedded system remotely from a desktop or, in my case, a laptop. Of course, you can also run the debugger on the console of the embedded system.

In conjunction with the GNU C compiler and related tools, you'll find commercial compiler and development environments. I listed one compiler vendor in my sources, and you can find others in the *Linux Journal*.

RUNNING SUPPORT

Although the recommended minimum size for running Linux with X-Windows is usually 8–16 MB, it's overkill for running Linux in an embedded system. What is the minimum configuration we need?

To run Linux on a PC architecture, we need at least an i386-class machine because Linux has to run in 32-bit protected mode. Several embedded '386 boards out there should run Linux just fine.

In fact, I was even able to boot and run Linux on my 16-MHz '386SX laptop with 5 MB of memory. It runs fine, although you probably don't want to compile a program or run an Xserver on it. But then, that's not the point of a minimum system.

The minimum memory configuration to load a standard Linux kernel and have it be able to do something is 4 MB. It should be possible to configure and build a minimum size kernel that should work in about 2 MB.

Now we have an i386 and 4 MB of memory, what else do we need? Well, to boot Linux, we need a boot device.

Linux can't use flash memory as disks, but it can boot from it. In fact, Linux can load and initialize the kernel using any kind of boot device the BIOS can handle. So, it's possible to boot Linux from a flash memory after all, but it can't access the boot flash-memory disk once it's loaded.

You may ask, what good is it to boot from flash memory if you can't access the

Listing 1—You need this minimum set of files to boot Linux and start an application program. Of course, you need to add more device-driver entries in the /dev directory if your application needs to access other devices besides the screen and the RAM disk.

```
/lib
/lib/libc.so.5
/lib/ld-linux.so.1
/bin
/bin/sh
/bin/insmod
/etc
/etc/ld.so.cache
/dev
/dev/console
/dev/null
/dev/ram
/dev/systty
/dev/tty1
/dev/tty2
/dev/tty3
/dev/tty4
/linuxrc
```

disk after booting? Well, there are at least two techniques you can use to initialize an initial RAM disk while booting, and this RAM-disk image can contain the essentials of what's needed to run Linux.

What do we need to run Linux once the kernel has booted? By incorporating our system into one Linux program, we can get away with a small set of files. Listing 1 has the details.

The most critical components are the console device entry `/dev/console` and whatever device entries that are needed. Once booted, the kernel opens and runs whatever program or command script is contained in `/linuxrc`. That's about all you need for a minimal configuration.

Even though Linux wasn't designed as an embedded OS, it ended up having features like small initial size and capability for RAM disks and modular device drivers. These features are there because the developers wanted Linux to be easy to install.

A single floppy can hold a complete mini-Linux system, complete with enough utilities to format and initialize the file system on a hard disk and do a bootstrap install of Linux from another medium. We can use this to our advantage by including our embedded applications, instead of the installation utilities to build an embedded Linux system.

Now let's look at a system I've been using to play around with embedded Linux. It's based on Motorola's embedded Pentium SBC, the NLX 55, shown in Photo 1.

The motherboard contains everything needed to build a system and run an embedded application. It has a 233-MHz Pentium MMX CPU, slots for two DIMM modules, 10-/100-Mbps Ethernet controller, dual-IDE controller, floppy control, and SVGA controller. It also contains a flash-card socket on the motherboard.

SanDisk's CompactFlash is a flash-based media that contains an IDE controller on the card itself. To the system, it looks just like another IDE drive. Motorola also sells a starter system that includes a case and power supply as well as a standard IDE drive for development.

With this system, you can develop embedded applications while running off the hard disk and build a flash-card module of our embedded system. Then, just reboot, change the boot sequence in the BIOS setup, and have the system boot from the flash card to test your application. Next month, I'll use it to build a flash card-based Linux installation.

INSTALLING LINUX

Obtaining Linux is easy. As I said, you can get most Linux distributions from the 'Net for free. All you need is patience while downloading it. You can also purchase a CD-ROM of various distributions. Most commonly available distributions have enough of the basic components necessary to allow you to embed Linux.

Once you have an installation, you should build a desktop system to serve as

<code>rtf_create(unsigned int fifo, int size)</code>	create a FIFO
<code>rtf_destroy(unsigned int fifo)</code>	get rid of the FIFO
<code>rtf_resize(unsigned int minor, int size)</code>	change the size of the FIFO
<code>rt_fifo_put(unsigned int fifo, char * buf, int count)</code>	write to FIFO
<code>rt_fifo_get(unsigned int fifo, char * buf, int count)</code>	read from FIFO
<code>rtf_create_handler(unsigned int fifo, int (*handler)(unsigned int fifo))</code>	create a handler that is called when a user process reads or writes to a FIFO device
<code>rt_task_init(RT_TASK *task, void (*fn)(int data), int data, int stack_size, int priority);</code>	create a task
<code>rt_task_make_periodic(RT_TASK *task, RTIME start_time, RTIME period);</code>	arrange so that the task is called at a periodic interrupt
<code>rt_task_delete(RT_TASK *task);</code>	get rid of task
<code>rt_task_wait(void);</code>	give up time slice, wait until next time slice
<code>rt_task_suspend(RT_TASK *task);</code>	suspend any task
<code>rt_get_time(void);</code>	read the current time
<code>rt_request_timer(void (*fn)(void));</code>	allocate a timer, which will call a handler when it expires
<code>rt_free_timer(void);</code>	return timer
<code>rt_set_timer(RTIME time);</code>	set the timer to go off at a specific time
<code>rt_no_timer(void);</code>	clear timer

Table 2—The real-time extension to Linux (RT-Linux) provides this API for programs that are written to run in this mode. Most of the services are pretty standard interfaces you'd see in a bigger RTOS. But, the FIFO interface can be used to communicate with non-real-time threads.



Photo 1—The Motorola NLX55 motherboard features a Pentium MMX CPU, graphics controller, and 10-/100-Mbps Ethernet controller. Unique to this board, it also includes a socket for a SanDisk flash-based ATA card.

your development platform. This process is relatively painless and there are a lot of resources on the 'Net. If your target system is capable enough to serve as a development environment, you can install it there.

RT EXTENSION OF LINUX

The standard Linux kernel doesn't pretend to be real time in the traditional way. Like many Unix kernels, the kernel is not preemptive and thus can't be relied on for predictable interrupt response. But, some clever people developed a real-time extender for Linux that turns it into RT-Linux.

RT-Linux has a layer added between the Linux kernel and the hardware timer interface. After installing the RT extension, the Linux kernel and its processes turn into a single real-time task that always runs at the lowest priority.

Real-time applications are loaded into the kernel memory space using the Linux module facility. Once the application has loaded and started real-time tasks, the tasks have control of all the hardware and memory in the system. RT-Linux provides calls to start, suspend, and destroy tasks. There's also a timer facility to set up timers with real-time responses.

A FIFO is used for real-time tasks to communicate with regular user processes. The FIFOs appear as standard Unix character devices that a user process opens and then reads and writes to. Table 2 shows the basic API available in the RT extender.

Although, RT-Linux is rather primitive in its implementation, it has the essentials you need to implement real-time applications.

WHAT'S NEXT

I've given you an overview of Linux and RT-Linux as well as how you can use it as an embedded OS. I'll concentrate on the details of Linux in upcoming articles.

Next month, I'll show you how to embed a minimal Linux configuration into a small system. After that, you'll learn more about Linux embedded software development and how to use RT-Linux.

You're also likely to see Linux in some of my projects. It may not be the answer for every embedded-systems problem, but it's another tool in the toolbox. And since it's still being developed and is constantly evolving, it will be interesting to see what problems people end up solving with Linux. [RPC.EPC](#)

Ingo Cyliax has been writing for INK for two years on topics such as embedded systems, FPGA design, and robotics. He is a research engineer at Derivation Systems Inc., a San Diego-based formal synthesis company, where he works on formal-method design tools for high-assurance systems and develops embedded-system products. Before joining DSI, Ingo worked for over 12 years as a system and research engineer for several universities and as an independent consultant. You may reach him at cyliax@derivation.com.

REFERENCES

www.linux.org
RT-Linux project at NASA, aol111.wff.nasa.gov/rtlinux/

SOURCES

C/C++/F77 compilers for Linux

The Portland Group
(503) 682-2806
Fax: (503) 682-2637
www.pgroup.com

NLX55 motherboard

Motorola Computer Group
(800) 759-1107, ext. PR
(512) 434-1526, ext. PR
www.mcg.mot.com

OpenLinux

(801) 765-4999
Fax: (801) 765-1313
www.caldera.com

RT-Linux

r52h146.res.gatech.edu/~bdixon/rtlinux

Linux distributions

RedHat
www.redhat.com

S.u.S.E., Gmbh
+49 911 7405331
Fax: +49 911 7417755
www.suse.de

Linux Journal

(888) 66-Linux
(281) 261-2581
Fax: (281)-261-5999
www.linuxjournal.com

Applied PCs

Fred Eady

emWare Top to Bottom

Part 1: Monitoring via the Internet

There may have been some good old days, but Fred's not necessarily one to live in the past. He's reaching into the future, and it's looking like the future relies heavily on control via the web. Join him for a look at emWare's offering.

Remember getting that first programmable calculator? Beat the crap out of that old slide rule, huh?

Remember that first contact with a BBS? The Internet makes all that look a little silly now, doesn't it?

Well, pretty soon, if not already, you will be making first contact with your first web appliance. If you keep up with the movies, all of the "first contacts" were pretty much unexpected. Some were catastrophic.

I'm going to change all of that here. A beginning look at what it takes to control and monitor your application via web browsers is right behind this paragraph.

Before I begin, I want to point out that there are at least two other products out there that are similar in nature to the one I'm going to discuss in this article.

The first—Phar Lap's HTML-On-The-Fly, which is included with the Embedded ToolSuite—is a unique implementation that enables you the programmer to assemble web pages as they are requested.

Phar Lap's version of web control comes wrapped within a very comprehensive development package that is now capable of using Bill's latest C++ compiler package.

Another web runner comes from Agranat. EmWeb's claim to fame is the elimination of the CGI and the melding of C and HTML. If you include EmStack, a TCP/IP stack, you don't even need an OS to implement EmWeb.

As you can see, the control-by-the-web marketplace is growing rapidly. I intend to explore as much of it for you as possible, but for now, let's concentrate on another contender in this area, emWare.

A 5000' VIEW

I really struggled coming up with a way to convey the new features of emWare's latest release. After a few days of reading and thought, I figured the best way was to show you what I saw on the screens and describe the code that was generated by interacting with those screens.

It's going to take a couple of passes, but by the time we realize our goal, you will have generated your own ideas as to how to integrate EMIT technology in your own projects. And, you'll garner enough basic EMIT knowledge to implement the package on your own.

The newest release from emWare takes advantage of today's object-oriented software technology to ease the emWare application-development process. Syman-



Photo 1—Everything's here. One CD does it all.

tec's Visual Café has been incorporated to make emWare's interface design much easier and more intuitive.

Also, instead of building emWare tables by hand as we did in the early days of EMIT, an upgraded package utility that uses a standard .ini file structure does all of that work for you.

And now, the developer can use Bill's Internet Explorer, as well as a plug-in-enhanced Netscape as end-user browsers. (I don't think the Justice Department had a say in this one.)

These are just a few of the enhancements in the latest edition of emWare. Let's get started by installing the new version of EMIT and all of its co-hosts. Photo 1 is where it all begins.

emWARE COMPONENTS

As you see in Photo 1, emWare uses Visual Café as a tool for integrating something physical and logical to something conjured like web browsers. Notice the first button, Install EMIT 2.5.

EMIT is short for embedded micro internetworking technology. That first word makes EMIT a topic of interest to us. Do what you will with the remaining words, but be aware that the "I" and "M" words are as just as important in the scheme of things as the "E" word.

By the way, the sign to the left of the install buttons alluding to the universal Ethernet is a pretty good clue as to what emWare and EMIT are all about.

Before diving into the install, I was not looking forward to hooking up to the Internet to download the browsers. It was a relief to see the browser-install buttons

when I fired up the toolkit CD. Somebody was on the ball with that one.

Let's start from the top down. Clicking the EMIT 2.5 button starts the install process. After the standard "who are you" screens, the dialog box in Photo 2 is presented.

Notice that you can install client and gateway code but no server code. Don't worry. That's OK. Let's climb back up to the 5000' level for a moment and I'll tell you why. emWare is an application set that consists of three major components: server, gateway, and client codes.

AT YOUR SERVICE

The server code—emMicro—is a tiny web-server application that resides on your target microprocessor platform. The target can be anything from an 8-pin PIC to a full-blown Pentium-based board.

For this discussion, my platform will be 8051 based, mainly because that's what ships with the EMIT SDK. There's no reason why an embedded PC or even a Microchip PIC couldn't be a target. In fact, once I finish with the hows and whys of emWare, I'll put this newly acquired EMIT knowledge to work on an embedded PC and PIC emWare application.

Judging from the target material, it would be logical to assume that the emMicro code is not lengthy. It's not. The server code is responsible for relaying variable information from the target server to the gateway device. As you'll see, this is all done via tables and a common telecommunications interface.

One of the most interesting aspects of emMicro is its use of microtags. Microtags are compressed references to interface

objects or device states. Normally, microtags are found on the server device and are expanded by the gateway device (with emGateway code) when needed. This process of tagging saves considerable space on the server machine and offloads some of the processing load to the gateway code.

All of this conversation between the server machine and the gate-



Photo 2—Depending on your application, you can load all of EMIT or just the pieces you need.

way machine is done via a lightweight network, emNet. emNet is responsible for handling the communication path between emMicro and emGateway.

Due to EMIT targeting the Internet, a serial link is the default method that emNet employs. For local implementations, standard Ethernet and RS-485 are also valid considerations. In a nutshell, emNet handles all of the low-level communications.

Functions, events, variables, and documents are all components of a complete emMicro server installation. Of course, the main user application is the boss, but it too has to be intertwined with the emMicro code to effect the EMIT functionality.

Functions are defined as the processes that the device performs. These functions are the standard jobs or procedures you code into any application.

Events consist of a table of predefined states that must be reported when that particular state occurs. An example of an event would be the tripping of a flood sensor in your basement.

Variables contain the state of physical or logical objects. For example, variables store the state of a logical switch or the numeric value of a thermistor reading.

Documents in this context are HTML pages that contain either information about the server device or HTML pages that are served on demand. In a sense, there's no real difference in functions, events, variables, and documents in the emWare environment than in any other program you would write.

The key here is that all of these program products are given the ability to be transported bidirectionally between the server machine and a web-browser application using the emMicro code.

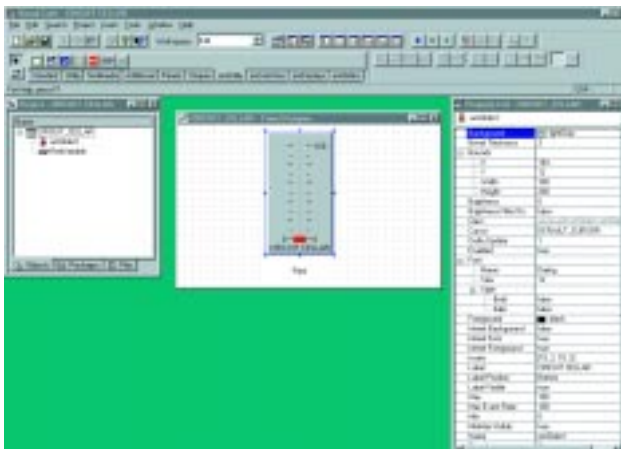
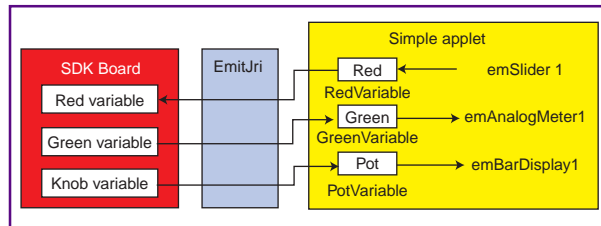


Photo 3—Hmm...looks a lot like Bill's Visual Basic.

Figure 1—EmitJri is the glue that binds the physical hardware to EMIT.



THE GATEKEEPER

Rather than siphon all that good server data into a bit bucket, emWare uses another component—emGateway—to provide Internet access and device management. The gateway code is respon-

sible for gathering and delivering commands and data to and from the emMicro server.

The gateway can also be configured to offload some of the service load from the server by acting as a repository for larger

pieces of data that can't be contained in the smaller server processor memory area.

emGateway normally resides on a host that is networked to the embedded server devices. If the server has enough resources, emGateway can be embedded there. The gateway code can also reside on the emClient machine. Where you put the gateway code depends on your available server resources and the network environment you're working with.

As with emMicro, emGateway contains building blocks that interface its code to other emWare components. Like emMicro, emGateway communicates at a low level using emNet. Each external server device must be managed by emGateway and this is done using the Device Link Module (DLM). The DLM provides port-management services for each external device's network connection.

A separate DLM codeset is used for each incoming device. If your application uses RS-232 and Ethernet servers, a DLM for each protocol and each device is called into action. The recruiter in this case is Device Access Service (DAS), which manages the external communication of data to and from the device and browser.

emGateway also contains the HTTP server that serves HTTP requests from the browser. Optionally, if some emObjects are not present at the browser level, emGateway can hold and serve these objects as well. These emObjects consist of HTML and Java objects not installed at the browser level.

If your particular application isn't browser-based, a command-line option is included in emWare to enable DAS to pass data via the DLM structure and emNet to and from the emMicro server.

I'm gonna push the stick forward and take us down a few thousand feet to take a peek at the client side of this operation. Visual Café-defined objects running in conjunction with a web browser constitute most of the client code. Again, as you will see a little later, control and monitor visual objects are defined and linked to the server through a Java applet.

The web-browser client is the means by which the user communicates with the emMicro server device. emGateway services are initiated via HTTP commands from the browser. The emWare-enabled browser uses emObjects and EMIT-specific Java files to effect this comm process.



Photo 4—Just think. This all used to be done “by hand.”

To be a full-fledged emClient, the emObjects and the Java-class files must reside on the local browser machine. Otherwise, if the emWare components are located on a remote emGateway machine, the client is no more than a standard web-browser configuration. Now that all the emWare players have been introduced, here's how it works.

A URL is sent from the browser to the emGateway device. Nothing is different about this URL. It contains address information needed to target a specific emMicro device in the network.

Typically, host, port, device, and document names are included in the requesting URL. emGateway's HTTP server receives the URL request and initiates a connection with the targeted server device.

At this point, the emMicro-enabled server device responds with the requested document. This document may be micro-tagged. If so, it's expanded by the microtag expansion services within the emGateway structure. If any of the necessary or requested emObjects aren't on the local browser machine, emGateway serves them, too.

Communication between emGateway's DAS and the web browser are arbitrated by EMIT's Java run-time interface (EmitJri). This link is established via HTTP info from emGateway. The client HTTP module is linked to the emGateway module for HTTP traffic, and a link is formed between the client's EmitJri interface and the emGateway DAS interface to transfer data to and from the server device.

Of course, the HTTP requests are logically linked with the DAS requests so that the two data conduits work together getting the requested documents from the emMicro server machine. Looks like the install can continue now, so let's move on.

As we descend, I've chosen to select all of these components offered because I

want to show as much code as possible.

Oops! The next panel states that I must have Visual Café installed to use the client development package. OK, I just exit the EMIT install and select the Visual Café install button. Eventually, I'll come back and finish where we left off on the EMIT 2.5 install.

COFFEE BREAK

Personally, the addition of Visual Café is one of the biggest improvements made

to the later versions of emWare. Before Visual Café was included in the SDK, all of the user interface and user interface code had to be done by hand with HTML tags and the like. Using Visual Café is a lot like using Visual Basic.

All of the objects are chosen from menu bars, and the properties of each object can be manipulated via a property sheet. Once the GUI objects are placed, Visual Café creates the appropriate code needed to define the objects.

Photo 5—Connecting the logical with the virtual physical is what this screen is all about.



At this point, the developer need only associate each object with a corresponding object on the emMicro server. emObjects can be switches, slide pots, analog or digital meters, or LED displays. emWare supplies the emObjects for Visual Café, and these are installed with the initial install of Visual Café.

An EMIT template and EMIT macros are also part of the initial setup and install process. The macros provide a mechanism for linking the display objects with variables on the server. Photo 3 is a look at the business end of Visual Café.

Take a good look at Photo 3 and let's walk through a slider setup. The first step is to click on the emSliders tab. Then, select a slider and drag it onto the Form Designer window. Select the slider by clicking on it and set the appropriate properties. That's all there is to putting a control on the panel.

Remember the emWare works on variables sent and received from the server application code. I must represent this variable just as it is represented on the server device. To keep it simple, I will assign a variable label and name it Red.

The variable label is used to communicate to and from the variables on the server device using EmitJri. I haven't defined EmitJri, so for now think of it as a conduit or path between the slider and its variable on the server.

Usually, the variable label is hidden, but for clarity I'll leave it on the Form Designer window. Once it is placed, I select it and set its properties. I'll call this variable Red-

Variable and set the text attribute to Red. From now on, this variable will be called RedVariable and Red will be displayed as the RedVariable label.

Now that the physical slider is defined and has been assigned a variable, it's time to link them. If you're wondering where Red is located and what the slider will control, I'll tell you.

The emWare SDK comes with an 8051-based platform that consists of a couple of LEDs, a pot, an RS-232 interface, and an EEPROM. It just so happens that one of the LEDs is red and its variable name is Red.

Now it makes a little more sense, right? OK, let's finish with the linkup.

After clicking on the slider I just placed and defined, I select Add Interaction. Just like Bill's stuff, a wizard is conjured up.

As you see in Photo 4, I selected RedVariable and instructed Visual Café to make my slider interact with it. After all is said and done in Photo 5, I told Visual Café to set up the code so that Red-

Variable would get its information from the slider control.

All that's left to do now is establish the path between the slider and the variable on the SDK board. This path is established by running one of the EMIT macros that was installed with Visual Café.

For my purposes, that macro is Add Embedded Event Listener. Photo 6 is what I saw before the command completed. Actually, I am setting up a link between EmitJri and RedVariable.

I should tell you a little more about EmitJri. This Java run-time interface between the applet resides on the client and the device access service on emGateway. The device access service manages the external communication of data between the server device and the browser.

The Listener macro produces Java code that enables the Red variable on the SDK board to get values from RedVariable in the client applet. RedVariable gets its values from the slider control. Figure 1 is a good logical view of the results of the actions I just performed.

With all that clicking and dragging, you might think there's code out there somewhere. Well, check out Photo 7. Everything I clicked on or dragged is now code.

You're probably wondering about going the opposite way through EmitJri like the Green or Knob variables in Figure 1. There's a macro for that, too—subscribe.

For simplicity, I didn't go both ways here, but for inquiring minds, subscribe is used when a device variable provides an update to EmitJri. The subscribe

macro provides Java code that enables any changes in the variables on the server to be automatically reflected in the client applet's emObjects.

Now that all of the code and GUI info for our simple applet is in place, the next thing to do is run it. Because I used the supplied SDK board and one of the predefined variables, this should be a snap.

The SDK firmware already has all of the necessary components needed to connect to the emGateway software. To create that link, all I need is to



Photo 6—As you would expect, the point and click generates code in all the right places.

start the emGateway application. Once emGateway is started and the SDK is powered up, I can execute my applet directly from the Visual Café application.

PREPARING TO PROGRAM

Although the emWare 101 course you just took was very rudimentary, I think you can see the possibilities that exist. emWare is designed to integrate into your application and provide a layer by which you can connect over virtually any TCP/IP-based network.

The basis to any emWare application is round-robin multitasking.

Simply put, this scheme gives all the parts of an application equal access to the processor resources. It means that no one part of the application should seize the processor resources for any abnormal length of time.

For example, let's say your application had to read some push-button switches to make a decision. If your program polled the switches in such a manner that the routine loops until a switch was released,

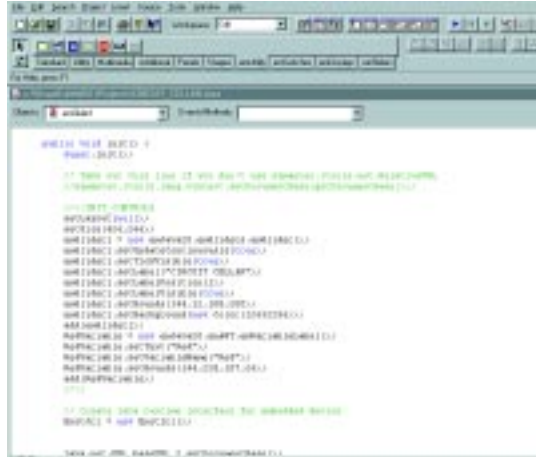


Photo 7—You want code? There it is.

then other parts of the program could lose or miss information they need to make their program-related decisions. If a program segment was waiting for incoming asynchronous data and the switch segment was looping...well, you can guess what happens.

PICING UP WHERE WE LEFT OFF

The final product of this discussion will be a PIC-based micro server attached to

an embedded-PC gateway serving a remote web browser.

I've introduced you to some of the emWare concepts at a high level. Next time, I'll drop to the tree-tops and talk turkey about what it takes under the covers to put our PIC on the network.

By the way, I did finish the EMIT install while you were looking at Visual Café. Thus, modern software technology from emWare has once again proven that it doesn't have to be complicated to be embedded. 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.

SOURCE

emWare
emWare
(801) 256-3883
Fax: (801) 256-9267
www.emware.com

DEPARTMENTS

64

MicroSeries

72

From the Bench

78

Silicon Update

Digital Processing in an Analog World

Technology Choices

MICRO SERIES

David Tweed

Part
2
of
3

This month, David discusses the pros and cons of several converter technologies in terms of performance, cost, and complexity. He covers everything from basic flash ADCs to PWM, so we know where to turn for our applications' needs.



In Part 1 of this series, I covered some of the basic concepts associated with analog-to-digital and digital-to-analog conversion, including quantization and sampling, plus some of the things that make a real-world converter deviate from the ideal performance.

This time, I take a look at several converter technologies and compare their strengths and weaknesses in terms of the performance characteristics I discussed, as well as other considerations such as cost and complexity.

I'll start with the conceptually simpler technologies such as flash ADCs and R-2R DACs and then move on through switched current, dual-slope, successive approximation, and pulse-width modulation. Along the way, you should develop a feel for where and why each technology has traditionally been applied and which one is best suited to your application.

FLASH ADC

The flash ADC is basically a brute-force implementation of the theoretical ADC, because for an n -bit converter, $2^n - 1$ analog comparators are used to compare the analog input directly and simultaneously to voltages representing the decision points.

The decision-point voltages are generated by a resistive divider of 2^n resistors connected between two voltage sources representing the lower and

upper limits of the desired conversion range. The basic structure is shown in Figure 1a.

The comparators generate $2^n - 1$ bits of information in what's commonly called thermometer code—all of the comparators below the input voltage produce ones, and all of the comparators above produce zeros. A table of this code for a three-bit converter is shown in Figure 1b. Logic is required to reduce this code to the n -bit binary code that is desired.

The logic to perform the conversion is quite simple, as you see in Figure 1c. Note that the middle column of the table is identical to the most significant output bit you want to generate. This bit splits the table into two halves, each of which contains a copy of a smaller table with half as many rows.

The middle column of this new table is the same as the next most significant output bit, and so on. Repeat this process until the last multiplexer is just one bit wide, and outputs the least significant bit. The output logic is a series of 2:1 multiplexers, each one controlled by the middle output bit of the previous stage.

The biggest advantage of this technology is its extremely high speed. If you put pipeline registers at the outputs of the comparators and multiplexers, the cycle time is limited only by the time it takes the comparators to settle and the registers to capture the outputs.

The data reduction takes $n - 1$ additional clock cycles for n output bits, but this affects only the converter's latency (delay), not its throughput. Flash converters usually operate in the 100-MHz to several-GHz range, providing between 6 and 10 bits of resolution.

The big disadvantage to flash converters is the circuit complexity. Because of the complexity and the heavy reliance on perfectly matched components, only monolithic (single chip) implementations make sense. For each additional bit of resolution, the number of comparators and latches must double, which drives up both chip area and power consumption.

The linearity of the converter is directly related to how well the individual resistors of the divider chain are matched. This relationship can be

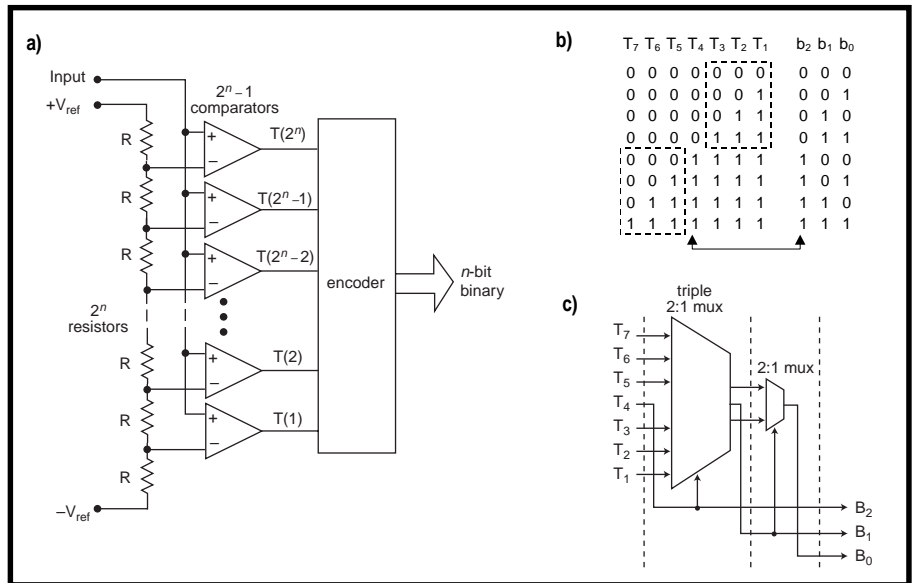


Figure 1a—Here is the block diagram of a flash ADC that directly implements decision levels. **b**—The thermometer-to-binary conversion table shows the symmetry that leads to the conversion logic. **c**—The conversion circuit is a series of multiplexers, with optional pipelining shown by the dashed lines.

mitigated somewhat by carefully laying out the resistors on the chip so that manufacturing tolerance issues (e.g., mask alignment) affect all of them in the same way.

Because this type of converter typically only makes sense in very high-speed applications, several dynamic issues come into play. One issue is the problem of distributing the input signal to all of the comparators.

The capacitive load of all those inputs is considerable, and any buffer amplifier that gets inserted into the path must be fast, stable, and capable of driving large currents. Also, the propagation delay of the comparators must be well matched, and the pipeline clock must be distributed to all of the registers with low skew.

As a designer of a system incorporating these converters, you should be prepared to deal with the huge volume of data they produce. This means wide buses running at extremely high speeds. Board-level signal integrity is of paramount importance.

Several gigasamples-per-second converters are available (typically used in DSOs). Triple 6–8-bit devices are available for TV-resolution video applications (typically 14.31818 MS/s [mega-samples per second]). Analog Devices announced a triple 8-bit, 300-MS/s device for high-resolution video displays that costs about \$25.

R-2R DAC

One of the simplest forms of DAC is known as the R-2R network, shown in Figure 2a. An n -bit converter can be built from $3n$ resistors of all the same value, making it quite attractive for low-cost discrete implementations.

To understand how the R-2R network operates, recall how a voltage divider (a voltage source and two resistors) is exactly equivalent to a circuit consisting of a single voltage source in series with a single resistor. This equivalency is known as the Thevenin equivalent of the original circuit. (There is also a Norton equivalent, in which the original circuit is replaced by a current source in parallel with a resistor.)

The Thevenin voltage is calculated by the normal voltage divider equation. The Thevenin resistance is calculated as the parallel combination of the two original resistors, as shown in Figure 2b.

Now, consider the first bit of the R-2R network, shown in Figure 2c. Depending on the state of the switch, this bit forms a voltage divider developing $0 \times V_{ref}$ or $\frac{1}{2} \times V_{ref}$. In either case, the Thevenin resistance is the same:

$$\frac{1}{0.5R + 0.5R} = R$$

If I now connect this equivalent circuit to the next bit, as shown in

Figure 2d, it's easy to see that this forms another voltage divider, again with resistance $2R$ on each side. The bottom end of the divider connects to either $0 \times V_{ref}$ or $\frac{1}{2} \times V_{ref}$, while the upper end connects to $0 \times V_{ref}$ or $1 \times V_{ref}$.

If you work out all the combinations, you find that the Thevenin voltage of this circuit is $0 \times V_{ref}$, $\frac{1}{4} \times V_{ref}$, $\frac{1}{2} \times V_{ref}$ or $\frac{3}{4} \times V_{ref}$. Once again, the Thevenin resistance is simply R . You can add as many bits as you like, and the output resistance will remain R , and the output step size will be:

$$\frac{V_{ref}}{2^n}$$

I've shown the drivers of the R-2R network as switches for simplicity, but they could just as well be a set of output bits from a microcontroller. The important thing is that the drivers should have much less resistance than $2R$. If not, their resistance should be well matched in both the low state and the high state and to each other, and the $2R$ resistors should be reduced in value to compensate.

The biggest problem with this kind of converter is matching the resistor values closely enough—you need tolerances on the order of 0.1% for a 10-bit converter. You also need to be aware of the output resistance of the bare resistor network, adding an output buffer when necessary.

One big advantage of this converter is that the output is a simple multiplicative factor times the input voltage, which itself can vary or even change sign if the drivers allow it. That

makes it useful in applications like digital volume controls and in op-amp feedback networks.

SWITCHED-CURRENT DAC

The switched-current DAC uses a technique that is best suited for monolithic (single-chip) implementations. A variation of the R-2R network or a circuit called a current mirror (and sometimes a hybrid of both techniques) is used to divide a reference current into ever-smaller streams. Then, electronic switches combine these streams to form the desired output value. Figure 3 shows the general scheme.

A reference current is supplied at the top, which gets successively divided in half. The output currents are switched onto one of two rails depending on the state of the corresponding bit.

Each rail carries the sum of the currents switched onto it, with the I_0 rail carrying a current proportional to the digital code and the \bar{I}_0 rail carrying the remaining current, which corresponds to the complement of the digital code. Each current needs to be returned to ground. Normally, the I_0 rail drives a current-to-voltage converter implemented with an op-amp, while the \bar{I}_0 is returned directly to ground, sometimes through a load resistor.

Because the currents are flowing through the current-dividing network in the same way all the time, regardless of the positions of the switches, it's relatively easy to keep them operating in a stable and accurate manner. Also, the voltages across the switches are very low and constant at all times,

so keeping them operating in a linear fashion is easy as well.

The key to good performance in these converters is that all of the transistors in the current mirror and switching currents must be carefully matched for their performance characteristics (e.g., threshold voltage and current gain). Also, they need to operate in the same environment as far as supply voltage and temperature, which makes monolithic fabrication the logical choice.

The main advantage of this kind of converter is its relatively low cost, which results from the relatively low complexity and small die size. Another advantage is its relatively high speed, controlled mainly by how fast the switches operate, especially when the desired output is a current value.

However, the high speed can turn into a disadvantage when an output voltage is desired, because the current source has a very high effective output impedance (an ideal current source has an infinite impedance). If there's any stray capacitance at the summing node of the converter, as shown in Figure 3, this becomes a relatively slow R/C low-pass filter that slows down the transition speed of the converter.

DUAL-SLOPE ADC

A dual-slope ADC operates by translating an unknown amount of current into a time period and then measuring it. The key element of this kind of converter is an analog integrator that integrates the unknown current for a known amount of time, and then the

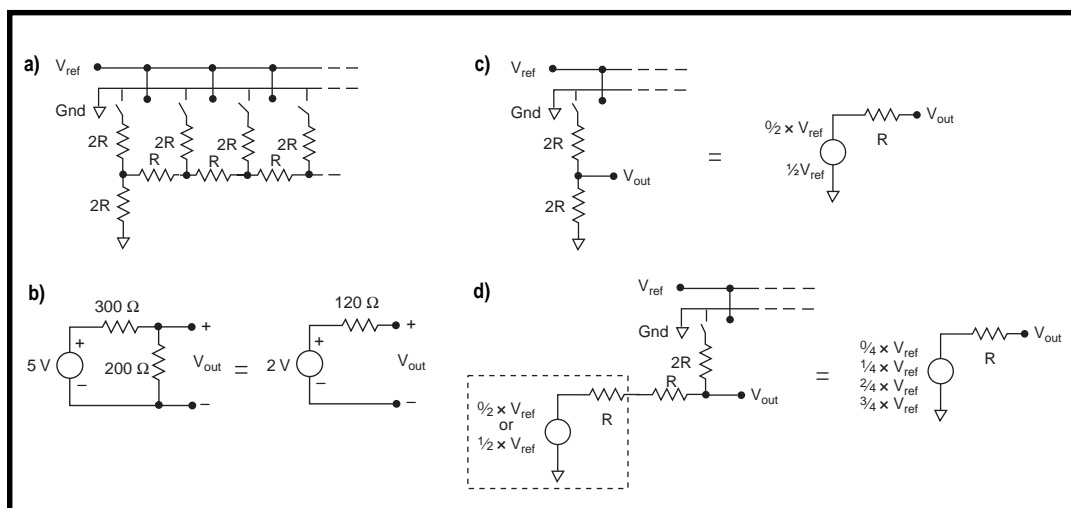


Figure 2a—The R-2R converter can be built from switches and resistors. **b**—A network of voltage sources and resistors can be reduced to a Thevenin-equivalent circuit. **c**—The equivalent of the first bit has a constant resistance R . **d**—Analyzing the second bit is easier using the Thevenin equivalent of the previous bit.

integrator is discharged by a known reference current while measuring the time required for the output to reach zero again.

A digital counter running at a constant rate measures the time period. An unknown voltage is easily turned into a current by applying it to a precision resistor. The block diagram of a dual-slope converter is shown in Figure 4a, and the corresponding timing is shown in Figure 4b.

The dual-slope converter operates in three phases. During the first phase, the switch SW_a is closed, which resets the integrator to zero voltage, and the digital counter is reset at this time as well.

During the second phase, SW_a is opened and SW_b is closed, applying the unknown voltage/current to the integrator for a fixed period of time controlled by how long it takes the counter to overflow.

During the third phase, SW_b is opened and SW_c is closed, removing the unknown current from the inte-

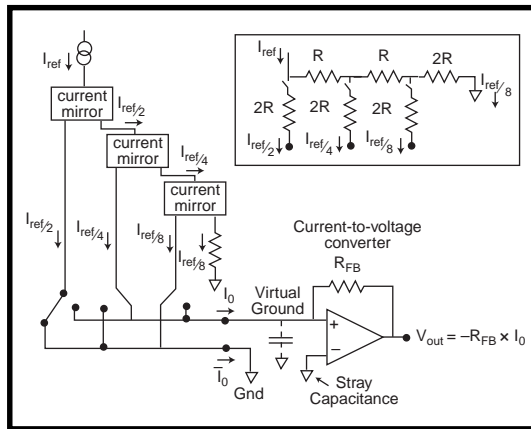


Figure 3—The switched-current DAC directs fractions of the reference current to one of two summing nodes. The inset shows how an R-2R network divides a reference current if all outputs are at ground or virtual ground.

grator and applying a fixed current of the opposite polarity. The counter starts counting from zero again, and its value is latched when the integrator's output reaches 0 V again.

The time for this process is directly proportional to the integrator voltage at the beginning of the third phase, which is directly proportional to the original unknown voltage or current.

A refinement of this circuit, known as the triple-slope converter, adds an extra phase between the first and second phases during which the integrator input is grounded and the capacitor is connected in the opposite polarity. This phase basically operates the integrator in the same mode as the original second phase but with the direction of integration reversed.

This phase also lasts as long as it takes for the counter to overflow. It allows the integrator to accumulate a charge that represents the offsets and leakage currents in the analog circuitry, which then cancels out those same errors in the next phase—the measurement phase.

When carefully constructed, this type of converter can yield very high resolution and accuracy. Also, the relatively simple structure requires relatively low power, which makes it suitable for battery-powered applications like digital multimeters.

So-called 3½-digit meters (2000 counts) are quite common and represent

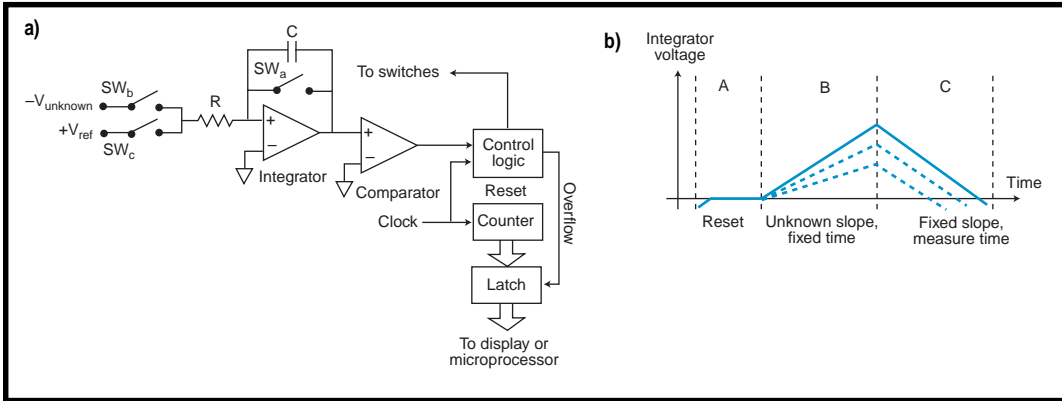


Figure 4a—The schematic of the dual-slope ADC shows the simple analog circuitry. **b**—The conversion is performed in three phases.

about 11 bits of binary resolution. Laboratory-grade instruments of six or more decimal digits (1,000,000 counts) are available and represent about 20 bits of binary resolution.

One disadvantage is the relatively long conversion time—on the order of hundreds of milliseconds—which allows only a few measurements per second. However, this conversion time is quite acceptable in a digital meter, and the long integration time has the beneficial side effect of filtering out most forms of noise.

One trick is to make sure the integration period is an exact multiple of 100 ms. Doing so makes the integration period an integer number of periods of 50 and 60 Hz, canceling out nearly all sources of power-line interference.

Parts are available with BCD outputs, which are preferred for direct metering, and binary outputs, which are preferred for microcomputer applications.

SUCCESSIVE APPROXIMATION

The difference between a dual-slope and a successive-approximation ADC

is the difference between doing a linear search versus a binary search—the time required is $O(\log(n))$, the number of bits, rather than $O(n)$, the number of counts. A block diagram of a successive-approximation ADC is shown in Figure 5a.

The concept is simple: The unknown quantity is applied to one side of a comparator, while a known quantity generated by a DAC is applied to the other. Digital logic monitors the results of the comparison and generates codes to feed to the DAC.

The algorithm divides the converter's range into two equal halves by initially setting the DAC to the middle of the range. The result of this comparison—a 1 if the unknown is in the upper half, a 0 if it is in the lower—eliminates half of the range from consideration.

The DAC is set to the middle of the remaining half of the range, and another comparison is made. As this process continues, the DAC's output becomes a better and better approximation of the unknown value. When the process is complete, the code then being fed to the DAC is the best approximation of the input value, and it becomes the output value of the conversion.

Figure 5b shows the timing sequence of a successive-approximation conversion. At the first step, the DAC is set to one-half full scale, and the comparator indicates that the input voltage is somewhere above that, shown by the shaded region. This setting also means that the most significant bit of the digital code will be a 1.

At step two, the DAC output is set to the middle of the upper range, and the comparator indicates the voltage is less than that. The second bit of the result will be a 0, and once again the DAC output is adjusted to the center of the narrower range. This process continues, subdividing the range containing the input value until all of the bits of the output code are generated.

It's important that the input voltage doesn't change during this process. Particularly if the voltage moves out of the range under consideration early in the sequence, it becomes impossible for the converter to come up with a correct code—the remaining bits will be all ones or all zeros as appropriate. This converter technology is often

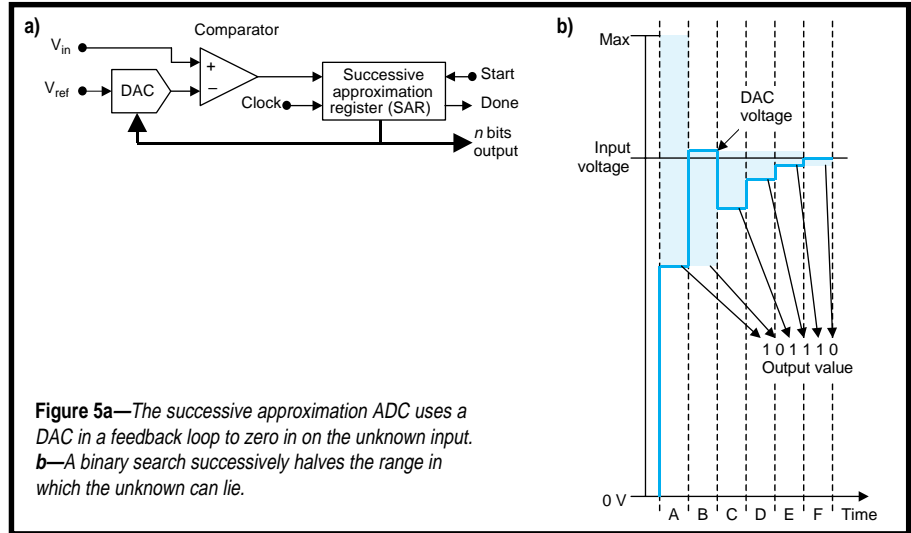


Figure 5a—The successive approximation ADC uses a DAC in a feedback loop to zero in on the unknown input. **b**—A binary search successively halves the range in which the unknown can lie.

paired up with a sample/hold circuit to address this problem.

PWM DAC

The pulse-width modulator (PWM) is a simple form of DAC with many positive features. It works on the principle that switching rapidly between two fixed values with a variable duty cycle and then taking the average over time can generate an analog value.

Figure 6 illustrates a PWM and the timing of its operation. A free-running digital counter feeds one side of a digital comparator, while the desired digital output code is fed to the other.

The comparator's output is high when the counter value is less than the code value. Higher codes mean you have high output for more of the time. One output pulse is generated for each complete cycle of the counter, so the sample rate is effectively the counter's clock frequency divided by 2^n , where n is the number of bits in the counter and the code word.

Even moderate sample rates and resolution can require high clock rates.

For example, 8-bit resolution at a sample rate of 8 kHz requires a clock of 2.048 MHz, and 16-bit resolution at 44.1 kHz (CD quality) requires a clock of 2.89 GHz! So, this type of converter tends to be used in low-speed applications (e.g., process control) in which conservative filters can be used.

The big advantage of the PWM is that the circuitry is nearly all digital, and the analog output filter is often nothing more than a passive R/C low-pass filter. These qualities enable the converter to be easily integrated with other digital logic, and in fact, many single-chip microcomputers are available with PWMs built in, making it the preferred DAC technology in extremely cost-sensitive applications.

Another advantage of the all-digital active circuitry is that the linearity of the converter is virtually guaranteed. The only sources of errors are the passive filter components, which tend not to have linearity problems, and the timing of the edges of the PWM waveform, which is easy to control.

One disadvantage is the fact that the raw output has a strong tone or carrier signal at the sample rate and at harmonics of the sample rate, and the output filter needs to have strong attenuation at those frequencies in order to eliminate them.

If a simple analog filter is going to be used, the sample rate must be much higher than the Nyquist limit would imply. Or conversely, the signal bandwidth must be kept much lower than half the sample rate.

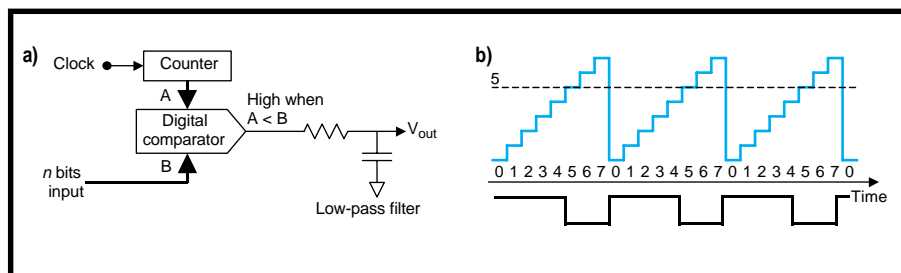


Figure 6a—A pulse-width modulator is nearly all digital circuitry. **b**—The PWM generates one output pulse for each full cycle of the counter. Showing the digital values of the counter (stairstep) and input code (dashed line) graphically makes it easy to see how the output waveform (lower trace) is generated.

CHOOSE A CONVERTER

I've covered a few basic converter technologies so you can choose the right one for your application. I'll wrap up by looking at delta-sigma converters and why they're the hot area of development these days. I'll also cover the whys and hows of using dither. 📧

David Tweed has been developing hardware and real-time software for microprocessors for more than 22 years, starting with the 8008 in 1976. His system design experience includes computer design from supercomputers to workstations, microcomputers, DSPs, and digital telecommunications systems. David currently works at Aris Technologies developing digital audio watermarking. You may reach him at dtweed@acm.org.

SOURCES

AKM Semiconductor, Inc.
(408) 436-8580
Fax: (408) 436-7591
www.akm.com

Analog Devices, Inc.
(781) 937-1428
Fax: (781) 821-4273
www.analog.com

Analogic Corp.
(978) 977-3000
Fax: (978) 531-7356
www.analogic.com

Burr-Brown Corp.
(520) 746-1111
Fax: (520) 746-7401
www.burr-brown.com

Cirrus Logic/Crystal Semiconductor
(512) 445-7222
Fax: (512) 445-7581
www.cirrus.com

Datel, Inc.
(508) 339-3000
Fax: (508) 339-6356
www.datel.com

Exar Corp.
(510) 668-7000
Fax: (510) 668-7001
www.exar.com

Linear Technology Corp.
(408) 432-1900

Fax: (408) 434-0507
www.linear-tech.com

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

National Semiconductor Corp.
(800) 272-9959
(408) 721-5000
Fax: (408) 739-9803
www.national.com

NEC Electronics, Inc.
(408) 488-6000
Fax: (408) 488-6130
www.nec.com

Philips Semiconductors
(800) 447-1500
(408) 991-5207
Fax: (408) 991-3773
www.semiconductors.philips.com

Texas Instruments, Inc.
(800) 477-8924, x4500
(972) 995-2011
Fax: (972) 995-4360
www.ti.com

Jeff Bachiochi

Part 2: The Show Must Go On



The show must go on, and so does

Jeff as he finishes his MIDI animator project. After discussing MIDI control, Jeff completes the performance by showing how lighting engineers can get in on the act.



Last month, we were learning a bit about the MIDI music control protocol and how a bit of hardware can capture the command sequences, when we suddenly smashed right into the end of the column.

Remember that the MIDI protocol defines events like note on/off, timing, and synthesizer presets to recreate performances that were recorded live or written as a musical score. Since both timing and note information is stored in a MIDI file, it seemed like a practical front end to a digital control system. Thus was born this project, the MIDI animator.

MIDI files can hold information for up to 16 instruments. So, this project has configuration jumpers for selecting which of the 16 instrument commands to listen to.

Next, only two commands are of interest—the 80H command for note off and the 90H command for note on. When one of the commands is decoded from the MIDI output's 32,768-Hz serial datastream, the following two bytes are saved along with the first.

The second byte indicates the note being turned on or off. Only 14 of the 128 possible notes are acted on. The remaining are tossed out.

The PIC16C63 has 14 outputs dedicated to these notes. When the MIDI

output commands any of these notes on, the MIDI animator turns the appropriate output on. And, when the MIDI output commands any of these notes off, the MIDI animator turns the appropriate output off.

The third byte of the command holds special information about that note, known as velocity. Here's where things get interesting. You can think of velocity as how hard a note is hit. If the note is hit softly, the third byte holds a low value. If it is hit hard, the third byte holds a large value.

In addition to the PIC's outputs going high or low, the velocity value can indicate an amount of high/low. This analog amount can be output in one of two ways—either PWM based on 60 Hz or a modified PWM used for servo motors. A configuration input determines the type of output used on each of the 14 digital outputs.

PWM

With most adjustable AC control, you have to be careful. If the control output isn't in sync with the line, there will be some modulation AC output at a rate equal to the difference between the line frequency and control output.

To eliminate this beating, all 14 outputs must be in sync with the line's zero crossings. One of the PIC's inputs (see Figure 1, *INK* 99, p. 78) is used as a zero-crossing input providing a rising and falling edge in which all the timings can be based.

Each AC cycle has two crossings, 8.33 ms apart. Therefore, all control must be done within that time frame. Since the velocity value will be 0-64, we'll break this 8.33 ms into 65 parts, with each part lasting 128 μ s.

The most common AC control device is the triac. The triac can be turned on anytime within each half cycle, and it turns off automatically at the next zero crossing (provided the gate control has been removed).

The velocity value received for any specific note (output) determines when (i.e., during which of the 65 cycles) the output will be switched high. The higher the value, the sooner it will be turned on.

When the value is 64, it remains on throughout all cycles. When the value

is zero, it remains off throughout all cycles. When the value is 32, it is turned on half way through each cycle.

Timer2 is used as the PWM timer. It restarts at each zero crossing so the timing stays in sync with the line. When Timer2 matches the period register (every 128 μ s), an interrupt is triggered.

Depending on how close to 60 Hz your line frequency remains, you can adjust this value to assure you remain at less than a half cycle for all 65 periods. Reducing this number means you can withstand more deviation in line frequency. However, the control at the minimum values will be a bit distorted.

The interrupt routine decrements a time frame count (starting at 65) and compares it with the velocity values for each output (held in a look-up table). If the values match, the appropriate output is set.

The outputs are reset after the sixty-fifth Timer2 interrupt (see Figure 1). Now the code just idles (but not for long) until the next zero-crossing edge and the pattern is repeated.

The circuit I used contains a power supply running off of 120 VAC. I put the circuit right on the board giving me access to AC, which enables sync with the line.

I used the change of state input to create the 120-Hz interrupt clock. If you don't need that, the system will use Timer0 as a 60-Hz source. The initialization routine looks at the zero-crossing input to determine whether to stay synchronized to the line or the internal oscillator.

SERVO CONTROL

Radio-control models gave us low-cost servo control. These servos' input is based on a modified PWM output pulse. The servo input must be 1–2 ms in duration. A 1-ms pulse is the minimum duty cycle, and a 2-ms pulse is a maximum duty cycle.

You can command the servo to move between minimum and maximum rotation by varying the duty cycle between 1 and 2 ms at a refresh rate of 30–100 Hz. Most servos rotate

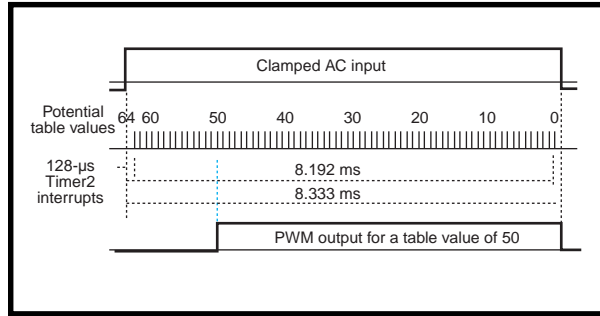


Figure 1—The upper trace shows one-half of a 60-Hz cycle. The middle trace illustrates the 65–128- μ s parts of each half cycle. The bottom trace shows a PWM output of the value of 50.

between extremes in tenths of a second. I chose to use the same cycle time as the previous PWM mode—60 Hz.

This task is accomplished by controlling pairs of outputs in sequence. It was a trade-off between time within the execution loop and waiting time for the next loop. Potential characters could come in every 300 μ s. Dividing 1 ms into 65 parts gave me only 15 μ s between each interrupt.

I needed to execute through the interrupt loop fast enough to allow plenty of time to process incoming characters and still get back for the next interrupt. I found that my interrupt execution took about 20 μ s, which left -5μ s and no time for the UART handler. I had to compromise.

I decided to divide the velocity value in half and only have 33 parts to the servo's modified PWM output. This decision gave me a 10- μ s overhead for the UART handler and still allowed a 60° servo control to a resolution of 2° (see Figure 2).

Timer2 is for the servo interrupt. When the zero-crossing interrupt hits, the first pair of outputs is set high and Timer2 is loaded with a value that will give the initial 1-ms output.

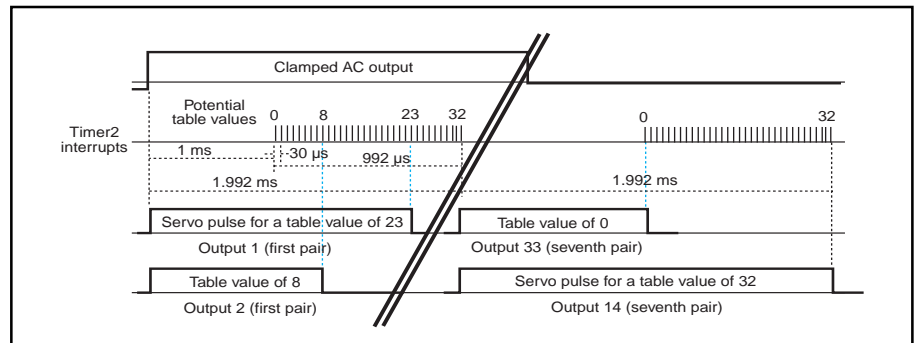


Figure 2—During each AC cycle (top trace), seven pairs of outputs are updated sequentially based on a 2-ms maximum time/pair. Servo outputs are on for a minimum of 1 ms.

When it overflows, it is reloaded with a value for 30- μ s interrupts. The next 33 interrupts check the velocity values stored in the note table. When the value in the table equals the interrupt count, the appropriate output is turned off.

After the thirty-third interrupt, the next output pair is turned on. The correct pulse output is created based on the table entries for these outputs.

In the meantime, the UART is being serviced and the note table is being updated. By the time all seven output pairs are processed, 16 ms have gone by and the next zero crossing starts the whole process all over again.

SERVO MECHANICS

Servos have three connections—power, ground, and control inputs. The internal circuitry uses a geared-down DC motor to turn an output shaft. A feedback potentiometer tells the motor-drive circuitry where the shaft is.

The motor-drive circuitry compares the pot's analog output to the duration of the input pulse. The motor is driven until the shaft turns the pot, so the two signals obtain equilibrium. Servos are complicated little modules when compared to their low cost. They are available for about \$15–50, each with torque on the order of 30–200 oz.-in.

Hardware included with the servo enables it to be used in almost any application where it either pushes or pulls through various linkages. You might use servos to animate robots, puppets, props, or kinetic artwork, and they can control many kinds of scale-model layouts as well.

LINE-SUNK PWM

Of course, the digital outputs of the MIDI animator can be used to control a variety of equipment. DC mechanical relays can be driven using a single noninverting open-collector buffer (i.e., 7407).

Many times, optocouplers are used with simple single-transistor output for speed or Darlington output for extra drive capability (see Figure 3). Solid-state relays are quite popular in eliminating mechanical bounce and contact pitting.

Most useful, however, would be proportional high-voltage control. Not only can solid-state SCR or triac drivers be used to turn high-voltage AC on and off, but PWM waveforms sunk to the line will give proportional control.

Although small-signal triacs can be driven directly from TTL outputs, most designers use an optocoupled driver to keep the AC and DC circuitry separated (see Figure 4). They're not only small and inexpensive, but they're also easily obtainable and second sourced by many manufacturers.

Standard triacs can require 50 mA of current to turn on. Optocoupled

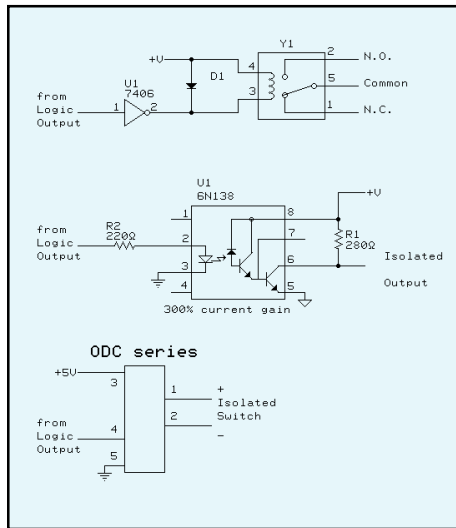


Figure 3—Here are three ways of controlling a DC voltage: mechanical relay, output transistor, or solid-state relay.

triac drivers (really sensitive gate triacs with limited current drive, usually a 100-mA maximum) are used to switch on larger triacs. Triacs are capable of controlling very high currents, and because of the power dropped across these triacs, heat sinking is a necessity.

Let's look at what happens when triac control is not sunk on the line. But first, there is a difference between solid-state relays (triac control with zero-crossing detection) and control that is sunk to the line. Solid-state relays with zero-crossing detection pay attention to the line voltage and only allow the relay to be turned on at zero crossings.

This characteristic prevents large in-rush currents and noise spikes from being produced when the control asks to turn the triac on at other than zero crossings. Fast $\Delta v/\Delta t$ edges are noisy. The built-in zero-crossing detector eliminates these by delaying the turn on until the next zero crossing.

The disadvantage of these devices is that they can't be proportionally controlled. To proportionally control the triac, we must be able to turn it on at any point throughout the cycle. The solid-state relay that has built-in zero crossing prevents this from happening.

Now, let's see what synching to the line gives us. The most obvious result is that we can perform the same kind of zero-crossing hold-off that is built into some opto devices. This ability is good for turning solid-state relays on and off quietly.

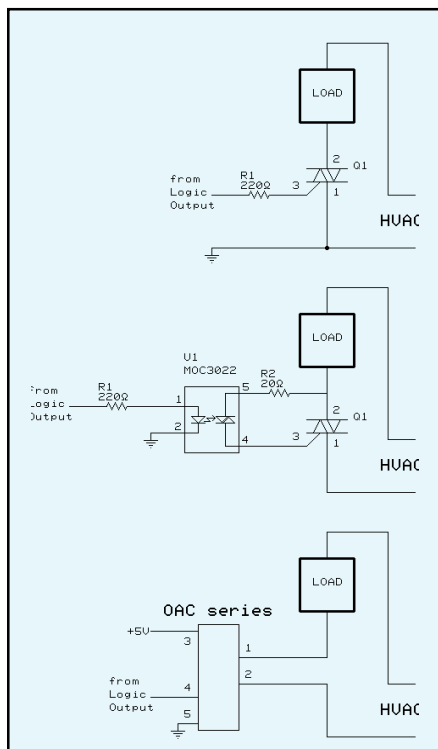


Figure 4—Here are three ways of controlling an AC voltage: nonisolated triac, optoisolated triac, and solid-state relay.

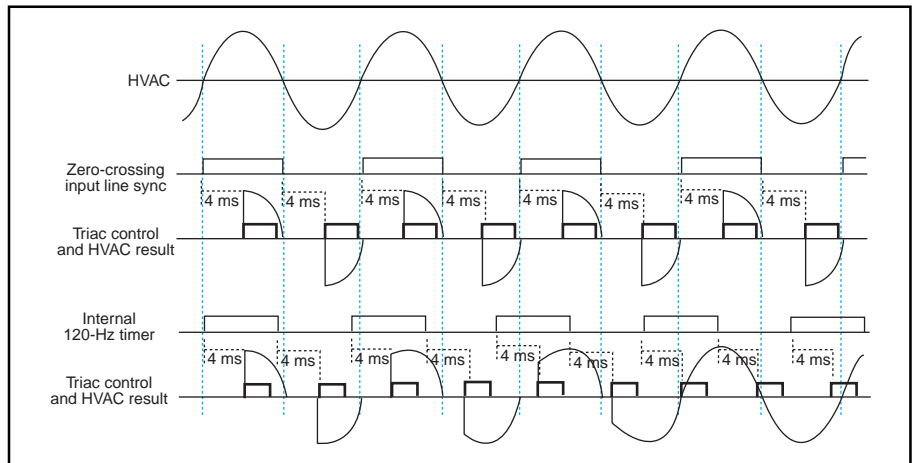


Figure 5—The second and third traces show a PWM output, which is in sync with the HVAC waveform. The fourth and fifth traces show how the PWM output no longer stays in sync with the line, based on a nonsynchronous oscillator.

But, the main use of this ability is to keep the proportional control signal in sync with the line, so the device is turned on at the same point in each cycle. Otherwise, you see a beat modulation as a pulsing of the output (see Figure 5).

For example, when the control loop is running at 55 Hz, the beat frequency is 5 Hz (the difference between 60 and 55 Hz). As the two become closer, the beat frequency drops, but if the two are not exactly the same, you get an annoying pulsing. Although this difference is mandatory for creating cool lissajous patterns, it's unwanted here.

ENCORE

To finish this project, I wanted to give the people who do stage lighting for concerts a chance to come into the spotlight (so to speak). So, I added opto triac drivers to each of the 14 outputs on the MIDI animator.

Each output drives a separate colored spotlight. The gel filters are arranged such that the lowest notes begin in the violet part of the spectrum, and the highest notes are at the red end. The notes in between correspond to their respective part of the spectrum. Finally, a synthesizer with MIDI output serves as the controller.

The lighting engineer can now join the band on stage, performing with shades of color that accent the music. Presuming the synthesizer's sound is off, the lighting designer can bang on that thing, mixing colors without having to worry about being tone deaf. That's real performance art, huh?

So, I've given you a way to play around with MIDI control using some software you may have received with your PC's sound card and some hardware you can easily build from scratch.

If you want more, check out MIDI show control. This spin-off is used for controlling theatrical and live performances, multimedia displays, audio-visual displays, and the like.

Although the format is quite similar to that of MIDI, tools for experimenting with it are expensive. Companies that manufacture MIDI show-control hardware often have their own tools. You can visit the many sites I've found to learn more about them. 📄

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.

REFERENCES

www.show-control.com/txt/mscspec.txt
www.harmony-central.com
www.midifarm.com/info/pcutilities.asp
ourworld.compuserve.com/homepages/jhuntington

SOURCE

PIC16C63
 Microchip Technology, Inc.
 (602) 786-7200
 Fax: (602) 786-7277
www.microchip.com

SILICON UPDATE

Tom Cantrell

Socket Rocket



The SX MCU from Scenix is fast, runs on almost

anything, gets good mileage, and you can park it anywhere.

And, Parallax's SX-Key makes for some easy riding. Will we be seeing biker bars in Silicon Valley?



Like other places, Silicon Valley goes through booms and busts. The only difference is that single-digit growth is considered a bust, while startups routinely post triple- or even quadruple-digit gains.

The boom of the last few years, fueled by I-way frenzy and bloatware-driven PC upgrades, has been one of the strongest ever (even though it's fading a bit in the stretch). It's a veritable gold (err...silicon?) rush.

One of the consequences is traffic with a capital "T". Ever fly into San Jose at 5:00 P.M., rent a car, and try to go somewhere?—anywhere? I don't commute, but I get out enough to see just how ugly the jam-up is.

THE NEED FOR SPEED

In my opinion, the ideal vehicle for our dart-and-weave stoplight-to-stoplight jousting is a motorcycle. Now, a bike isn't suitable for everything—certainly not big jobs like hauling the brood around or moving furniture.

I also understand the objections from those of you who live in colder climes. And, there's the danger factor, though two-wheel advocates argue that slow mummification behind the wheel isn't a real pleasant way to go either.

I rode a bike years ago when I was a commuter for the same reasons I give today. They're relatively inexpensive

to own and operate, they're allowed to use the carpool express lanes, and you can always find a parking space.

But, the most compelling advantage is speed! At the time, my 750 was faster off the line than practically any car on the road.

The inspiration for this thread is the appearance of the SX MCU from startup Scenix Semiconductor. Like a bike, it's small, relatively inexpensive (under \$4 in small quantities), and most important, fast. I'm talking close to 50 MIPS, which is a good 10–50 times faster than typical 8-bit MCUs.

EASY RIDER

It's no problem finding parking for the SX, which is offered in a variety of small packages from 18 to 28 pins, including DIP, SOIC, and SSOP. It isn't finicky about fuel either, running on anything between 3.3 and 6.25 V. It gets good mileage, consuming about 1 mA/MHz at full throttle and mere microamps when idling.

Take a close look at the motor (see Figure 1). Those of you familiar with Microchip PICs will notice a striking similarity. Indeed, the SX goes out of its way to offer PIC16C5x socket compatibility.

The novel in-system programming scheme (using the OSC pins) for the onboard 2K × 12 program flash is supplemented with a parallel programming mode similar to the PIC's, with a slightly different algorithm.

It's possible to configure an SX as a PIC clone, but it probably doesn't make much sense. Although it's not expensive, there's no way an SX is going to compete price-wise with the much higher volume PIC.

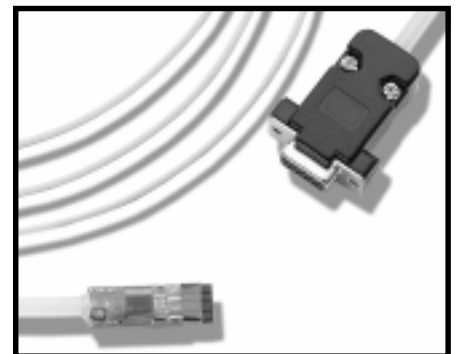


Photo 1—Like the SX, the SX-Key development tool from Parallax packs a lot of punch into a small package.

In fact, after considering the plethora of good 8-bit micros (including flash-based) on the market, two features stand out as compelling advantages for the SX—performance and innovative, low-cost, easy-to-use development tools.

ZIPPING IN AND OUT

The secret to SX performance is simple, relying as it does on the traditional technique of pipelining. The four-stage pipe in Figure 2 is a classic, similar to those found in earlier (but typically larger, like 32-bit) machines.

There is one, and only one, reason to use a pipeline and that's to boost the clock rate, which ultimately is limited by memory access time.

In compatibility mode, the SX reverts to four clocks per instruction (eight for JMPs and CALLs), same as a PIC. Flip the turbo switch, and the pipeline kicks in.

Once filled, the pipe delivers close to one instruction per clock. However, as with all pipelined machines, there are some caveats to be aware of.

The JMP and CALL penalty is relatively worse due to the need to refill the pipe. Where such instructions require two cycles (eight clocks) in compatible mode, they need three cycles (three clocks) in turbo mode, derating the advantage to 2.66x (8 divided by 3) for those instructions versus 4x for most others.

Another example is IREAD, one of the ten new instructions added by Scenix (see Table 1). IREAD enables a program to read the instruction memory, something that's nontrivial in a

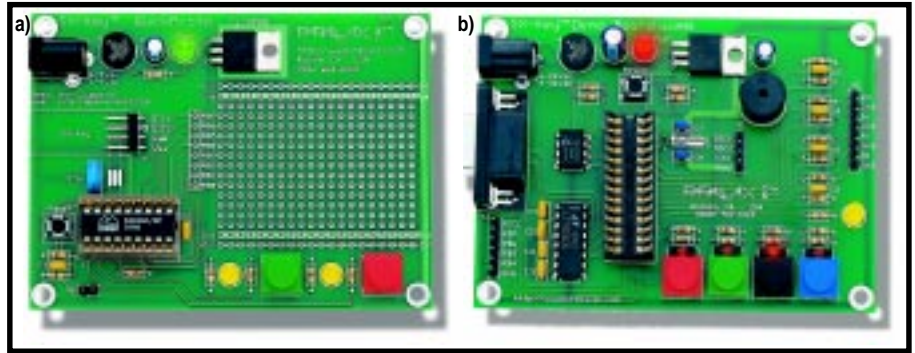


Photo 2—Included in the Master Key packages, Parallax offers a QuickProto board (with Master Key 18 package, or \$69 separately) and DemoBoard (with Master Key 28, or \$99 separately).

Harvard design (separate program and data memory).

Given the complication involved, IREAD requires the same number of clocks (four) in both compatibility and turbo mode. But, it's faster than previous data-lookup schemes and can access the entire code space.

Pipelined machines are also subject to various hazards that must be obviated by hardware, software, or both (e.g., the problem of trying to read data at the same time it's being written).

Consider a sequence of instructions involving a back-to-back write followed by a read of the same data. Instruction n is writing data (write stage) even as instruction $n + 1$ (execute stage) wants to read it.

With on-chip RAM, the SX includes forwarding logic that handles such an obstacle transparently in hardware. Thus, one instruction can write to RAM and the next one can safely read from the same location. However, for I/O ports, there are precautions concerning successive operations.

For pins configured as outputs, the SX reads the actual pin level, not the output latch. I think the SX approach is superior because it enables the detection of external problems such as a shorted or excessively loaded pin.

It's easy to confirm that the output-pin level is or isn't what it's supposed to be. By contrast, reading the output latch, rather than the pin, leaves you blind to outside interference.

As a consequence, a write to a port may not propagate through to the pin in time to be recognized by an immediate read. Depending on the clock rate and pin loading, a non-port instruction should be inserted to split up a back-to-back port write and read. Similarly, the possible difference between output latch and pin level calls for care when using read, modify, and write instructions like SETB and CLRb.

The I/O pins themselves (4-bit Port A, 8-bit Port B, and, for 28-pin devices, 8-bit Port C) are versatile. Each pin is individually programmable as input or output, with or without an internal pull-up resistor. All inputs are selectable as TTL or CMOS levels, and Port B and Port C inputs can be individually defined as Schmitt triggered.

Outputs can sink and source 30 mA (subject to overall device power limit), with those on Port A featuring symmetrical drive (i.e., centered about $V_{DD}/2$ under any load). This feature is useful for driving speakers and other pseudoanalog functions such as using a PWM to implement a DAC.

As inputs, pins of Port B can be individually enabled to act as wakeups (with programmable edge selection) from low-power sleep mode. Or, three pins of Port B can be configured as an

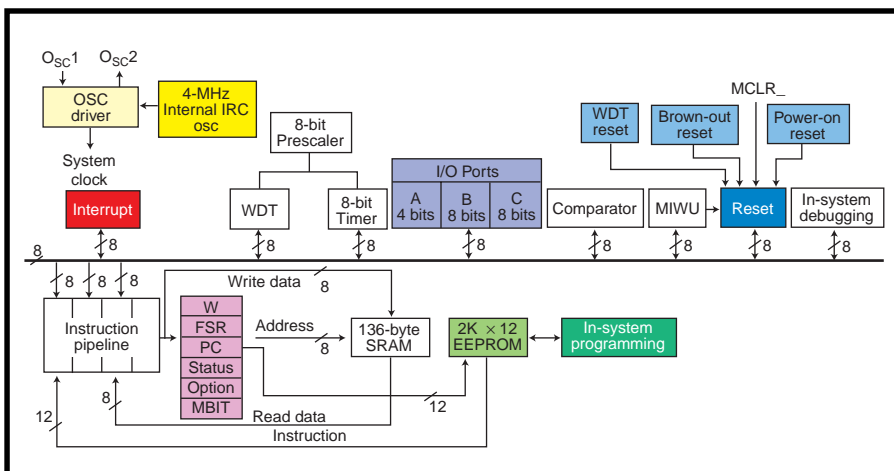
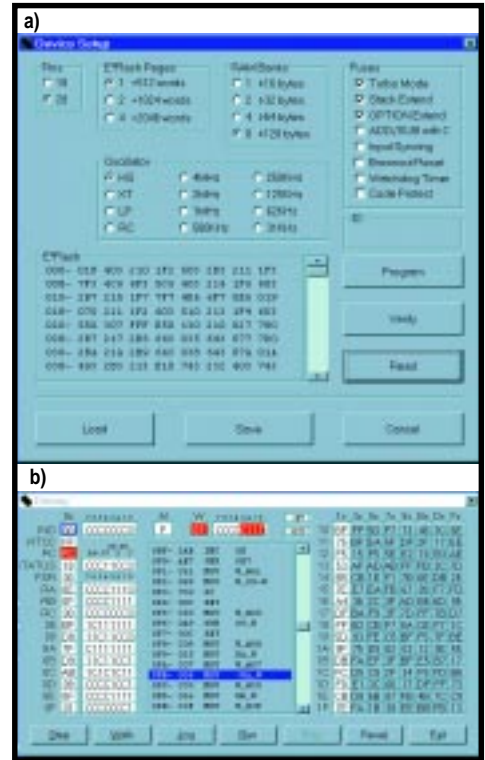


Figure 1—Like a motorcycle, the Scenix SX may be small, but at close to 50 MIPS, it's really fast.

Photo 3a—The Parallax SX development software handles configuration and programming of the chip. **b**—It also handles debug of ASM programs. Notice the Parallax-defined mnemonics and multiword macro instructions.



analog comparator. Two inputs (RB1 and RB2) are compared with the result (greater than or less than) reflected on output RB3.

Besides general-purpose I/O, the SX includes an 8-bit timer/counter (RTCC) and watchdog timer, either of which (but not both at the same time) can be mated with an 8-bit prescaler.

THROTTLE RESPONSE

The inertia of a full-size four-wheeler isn't a good match with stop-and-go traffic. The only winners are OPEC and brake shops.

The same goes for chips and interrupts. The grander the CPU, the more energy and time wasted finishing off instructions in progress, saving a bunch of registers, and making the turn toward the handler. Here, bikes—and the SX—have a big advantage. Less iron to stop and get going again means a quick and efficient response.

Interrupt sources include the RTCC and pins of Port B configured for wakeup (if the SX isn't sleeping, a wakeup functions as an interrupt).

Most CPUs require the instruction in progress to complete before anything else happens, but not the SX. To cut response time to the bone, the SX aborts instructions in the pipe. It also includes a set of shadow registers that automatically capture the critical state.

The end result is a blazing interrupt response: only three clocks for an RTCC interrupt and five clocks for external interrupts. That's 60/100 ns (RTCC/external) at 50 MHz. Thanks to the shadow registers, returns from an interrupt are equally speedy at three clocks.

GRAB THE KEY, HIT THE ROAD

Another nice thing about bikes is they're easy to work on, requiring

only a small set of basic tools. It's the same with the SX, thanks to the SX-Key from Parallax.

This nifty gadget, shown in Photo 1, exploits the fact that the SX has programming and debug logic onboard, accessed via the OSC pins. In-system programming and debugging for any SX-based design is a simple matter of incorporating a four-pin header (OSC1, OSC2, power, ground). For programming, the OSC pins act as a serial download channel, while during debug, the SX-Key has explicit clock control.

Three packages are offered for your riding pleasure. The Skeleton Key (\$249) is just the SX-Key and software tools. The Master Key 18 (\$319) adds a proto-board with buttons and LEDs (see Photo 2a). The Master Key 28 (\$349), shown in Photo 2b, comes with a fully loaded demo board including buttons, LEDs, RS-232, an external EEPROM, and a speaker.

On the roads these days, there's a remarkable proliferation of SUVs. Folks seem to think they need a cross between a Humvee and a Mack truck to get across town. It's kind of equivalent to the bloatware phenomenon that curses our PCs.

Command	Description
PAGE <i>n</i>	Switch to new instruction page
BANK <i>n</i>	Switch to new register bank
RET	Return without affecting <i>W</i>
RETP	Return across page boundary
RETI	Return from interrupt and restore state
RETIW	Return from interrupt and reload timer
IREAD	Read instruction memory
MOV !M,W	Write I/O mode bits
MOV W,!M	Read I/O mode bits
MOV !rx,#imm	Write immediate to port control register

Table 1—To ease programming and accommodate new features, Scenix adds 10 new instructions for a still RISCy grand total of 43.

Not so with the SX-Key software (Win 95 and up), which is blessedly simple, boiling down to three screens: one for editing your ASM program, one for configuring the SX, and the debug screens in Photo 3.

The set-up screen offers a good opportunity to top off the SX feature list. You can see the variety of clock options (crystal, resonator, RC, and internal 4 MHz with an eight-stage divider), configurable brown-out reset, extended stack (eight levels versus the usual two), tweaked operation of the

carry bit, optional input synchronizers (metastability insurance at high clock rates), and so on.

The assembler uses the Parallax mnemonics popularized on their earlier generation of PIC tools (there is a utility available to convert existing PIC code to the SX/Parallax mnemonics). Also, Parallax has defined a number of convenient macro-instructions that consolidate common sequences of single-word instructions into easier-to-use formats.

For those of you so inclined, there's a C compiler from Byte Craft. Data-type support is impressive with 8-, 16-, 24- and 32-bit INTs, and even IEEE-754 floating point, though I suspect it's easy to bite off more than a 2K-word SX can chew.

It's not cheap (\$795 DOS, \$1495 Win 95/98/NT), but I've always found that C compilers embody that old axiom "you get what you pay for."

One open issue involves integration of the Byte Craft and Parallax tools. At the time of this writing, they aren't really connected. However, both companies are planning a cozier relationship.

SX APPEAL

The conventionally wise will proclaim the concept of a high-performance 8-bit MCU as an oxymoron. It's true that the majority of apps are, and will continue to be, well served by the few MIPS most MCUs can muster. Yep, don't expect to see your average commuters crowding the Harley dealer, either.

Fact is, I think Scenix might be onto something. With performance to burn, the SX handles many functions in software that might otherwise call for extra silicon.

The concept of replacing hardware with software (Scenix refers to virtual peripherals) isn't new. In fact, some of the earliest micros were used to just such an end. Witness the 8048 called into duty as a keyboard encoder in the original PC.

The difference is that the SX performance headroom boosts the specs of the usual peripheral functions like serial I/O, software timers, and waveform generation. More speed, more channels, or both.

For instance, consider the well-known hack of building a UART in software. Transmitting is easy since timing is under the control of the programmer. But, reception is another story. Few apps can afford to dedicate all bandwidth to polling the serial line, so interrupts are required.

Thus, the achievable data rate is directly related to interrupt latency, which should be less than half the bit time. For instance, most 8-bit MCUs can handle 9600 bps, which enables a leisurely 50+ μ s from the leading edge of the start bit (that generates the interrupt) to the first sample.

But, higher data rates are a dicey proposition. For instance, 115 kbps needs an interrupt response of less than 5 μ s. Speed not only becomes an issue, but jitter (i.e., nondeterminism) is a factor as well. Consider the 8051, which can only guarantee response somewhere between 36 and 108 clocks, or 3–9 μ s at a 12-MHz clock rate.

To be fair, much of the variance is because the '51, like most MCUs, requires completion of the instruction in progress, including the relatively slow (48 clock) MUL and DIV, which the SX doesn't have. Banning MUL and DIV would cut 24 clocks (2 μ s at 12 MHz), making a worst-case response of 7 μ s—still not fast enough.

It's worse than it sounds because I'm just talking about the raw interrupt response. More time is needed to save critical state (PSW and any other registers), not to mention the instruc-

tions required to sample, compare, flag an error (false start bit), and so on.

By contrast, running at 50 MHz with five-clock external interrupt response (i.e., 100 ns) makes 115 kbps seem like a leisurely Sunday ride. I suspect you could hit 1 Mbps before going to full throttle. When it comes to interrupts, the SX blows the pins off regular MCUs.

The extra bandwidth doesn't mean doing old stuff faster. It lets the SX accept the challenge of significantly snootier I/O (e.g., music and voice synthesis and software video generation).

Scenix recently announced the availability of a Bell 202/CCITT V.22-compatible 1200-/2400-bps modem implemented in software, including not only the raw signal processing (FSK/DPSK) but also all the accessories (DTMF generation and detection, call progress detection, and caller ID).

Yeah, you can stick to your regular 8-/16-bit econo-chip for basic transportation, but remember that the fast lane is for passing. When that dot in the mirror turns into an SX hugging

your tail, you've got two choices: move over or get run over. ☠

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.

SOURCES

SX MCU

Scenix Semiconductor, Inc.
(408) 327-8888
Fax: (408) 327-8880
www.scenix.com

SX-Key

Parallax, Inc.
(916) 624-8333
Fax: (916) 624-8003
www.parallaxinc.com

C compiler

Byte Craft Ltd.
(519) 888-6911
Fax: (519) 746-6751
www.bytecraft.com

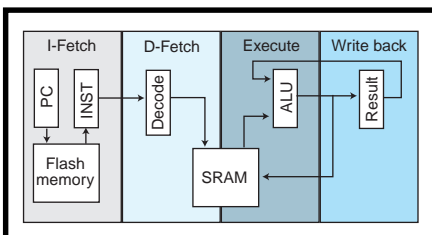


Figure 2—SX performance is obtained via pipelining using a time honored fetch-decode-execute-writeback design. Since a pipeline can only run as fast as its slowest stage, great attention was paid to the flash-memory design to achieve the 10-ns access time required by the 50-MHz clock rate.

PRIORITY INTERRUPT

What's PC?



One issue that I always wrestle with around here is separating hype from reality in the things we publish. My first thought when we review an article for publication is always, is it real?

Up to now, my test for "real" applied to the physical electronic components and usually never to the development platforms or the computers running the application.

In my own writing, I went to great pains to maintain cross-platform compatibility. My early *BYTE* articles always interfaced through a parallel I/O port and frequently ran in high-level languages. There were many brands of computers. Some had dozens of I/O channels and supported many other languages, but they all seemed to have at least one parallel port and BASIC or C. Later, as the technical world standardized on the PC architecture, I adopted it as my universal development tool and application platform.

Today, it should be even less of an issue. Everyone has a PC. I should be able to immediately assume that any article employing a PC is usable. Unfortunately, these days, I am having trouble defining exactly what a PC is and what configuration I should assume all of you think it is.

The long ball and chain connected to the past has severely hindered PC hardware and software design over the years. It's a wonder this computer architecture still works given the plethora of design iterations it has had to endure. I mean, look at it. Even with Windows 98, after 18 years, we still have the ISA bus, a 16-IRQ interrupt limit, and a silly 1-MB memory barrier in real mode. Heaven forbid, any of you reading *Circuit Cellar INK* presumes that an ultra-fast Pentium running a "real-time" process control program under Windows 98 is even remotely real time. About the only thing that seems to have advanced in parallel with its developmental need is graphics. Is this because we all play computer games, or was there a grander goal in mind?

Another issue is I/O interfacing. Take a look at the mess hanging off the back of your PC. Just how many peripherals can you attach to two serial ports and a parallel printer port anyway? If your computer is anything like mine, there are a couple switch boxes and a web of cables.

Of course, we've had innovations that were intended to solve this dilemma. Remember the SCSI bus? I have an HP scanner sitting at the end of a SCSI cable next to my computer. Of course, I never had another peripheral that shared this SCSI bus that didn't have to be internally mounted because of noise or wasn't better accommodated through an IDE connection instead. I still haven't figured out where I'm supposed to find something I want to use with IrDA yet.

Software and hardware designers have proposed many new standards to take the PC into the new millenium, including the Universal Serial Bus (USB), FireWire, and more modern operating systems like Windows NT and Linux. But, these new innovations have been a hard sell. Perhaps it's simply the mentality that if it ain't broke, don't fix it. Computer purchasers are resistant to the radical price changes that typically follow revolutionary technical innovation, and hardware manufacturers don't want to get caught out on a limb offering a product line that's overpriced or unsupported. Consequently, we end up with the vast majority of PC sales being ever-lower priced versions with the same, albeit higher speed, ingredients. Without innovation in the hardware, how is the industry to move forward?

It's definitely going to take guts from someone to take the initiative. Such innovation is something that should be coming from Intel, but I believe Intel has such a vested interest in the past that it becomes a disincentive for it to break from tradition. Microsoft is a powerful company and I suppose it could unilaterally impose software changes that dictated hardware architectural changes to implement them. Such fanciful wishes ignore the Wintel legacy, however, and Microsoft's vested interest in the past as well. Of course, I don't know any other company that has succeeded so completely at convincing customers to pay for the aggravation of changing what ain't broke on their PC. And, since virtually all the prior operating-system changes seem to obsolete the hardware anyway, it shouldn't be a problem persuading users that any new OS just involves changing it a little more.

I wish I had a crystal ball, but I don't. Undoubtedly, it will take computer vendors like IBM, Hewlett-Packard, and Compaq to break with tradition and take the high ground in the next generation of chip development. I trust that the embedded-control industry will follow their lead. Whatever the outcome, however, you can count on my continuing to ask "what is real?" when it come to the articles we present.

A handwritten signature in black ink, appearing to read "Steve Ciarcia".

steve.ciarcia@circuitcellar.com