

SPECIAL BONUS SECTION: HOME AUTOMATION & BUILDING CONTROL

CIRCUIT CELLAR[®]

THE COMPUTER APPLICATIONS JOURNAL

#54 JANUARY 1995

EMULATORS & SIMULATORS

Embedded Debugging
Using One Wire

Simulating Digital Filters

ARM Processor
Software Development

Decoding Bar Codes



\$3.95 U.S.
\$4.95 Canada

EDITOR'S INK

It's Only Skin Deep



y now, I'm sure you've seen our new outward appearance and have probably frantically flipped through the rest of the magazine looking for anything else we may have mucked with. Not to worry. Our content

hasn't changed one bit. We'll continue to bring you the kinds of practical hardware and software articles you've grown accustomed to.

Also new in this issue is the first of our quarterly special inserts called *Home Automation & Building Control*. We start the section with an overview of how the coax and telephone cable you probably already have in your house can be used to network your AN equipment with your personal computer. Author David Gaddis has authored numerous books and videos on home automation and has been close to the industry for years.

Next, Steve gets back to basics with a look at how to make hard-wired connections to the HCS II home control system. While the HCS is used as an example, the ideas can be applied to most any home controller that supports hard-wire connections.

From wires, we go to wireless and design a layout for a hand-held infrared remote control. This one not only can be used to send commands to AN equipment, but can send complete programming sequences to a whole-house controller. Such a layout isn't as easy as it first might seem.

Finally, turn your PC into a telephone attendant with a project that uses the newest in digital answering machine technology. No longer will unwanted telephone calls interrupt your dinner or evening entertainment.

On to our regular features, we kick off 1995 with a look at what's involved in rolling your own software simulator. There is no better way to get to know the processor you're using than to simulate its operation right down to the last status flag.

Once you've moved the code onto the target hardware, what about a technique that lets you get into the processors head using just one output bit? Our next feature describes this nifty technique.

Back to simulation, what about using a financial spreadsheet to help design a digital filter? It really does work and can be quite useful.

Finally, we wrap up our series on the ARM processor by covering some software tools that ease code development for the chip.

Briefly looking at our columns, Ed continues his protected-land journey by starting to learn how to juggle more than one task at once, Jeff develops a micro-powered wake-up circuit for low-power data loggers, Tom checks out the latest 8051 improvement dubbed the XA, and John implements a simple bar-code reader.

CIRCUIT CELLAR®

THE COMPUTER APPLICATIONS JOURNAL

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

PUBLISHER
Daniel Rodrigues

EDITOR-IN-CHIEF
Ken Davidson

PUBLISHER'S ASSISTANT
Sue Hodge

TECHNICAL EDITOR
Janice Marinelli

CIRCULATION COORDINATOR
Rose Mansella

ENGINEERING STAFF
Jeff Bachiochi & Ed Nisley

CIRCULATION ASSISTANT
Barbara Maleski

WEST COAST EDITOR
Tom Cantrell

CIRCULATION CONSULTANT
Gregory Spitzfaden

CONTRIBUTING EDITOR
John Dybowski

BUSINESS MANAGER
Jeannette Walters

NEW PRODUCTS EDITOR
Harv Weiner

ADVERTISING COORDINATOR
Dan Gorsky

ART DIRECTOR
Lisa Ferry

GRAPHIC ARTIST
Joseph Quinlan

PRODUCTION STAFF
John Gorsky
James Soussounis

CONTRIBUTORS:
Jon Elson
Tim McDonough
Frank Kuechmann
Pellervo Kaskinen

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 (203) 875-2751. Second class postage paid at Vernon, CT and additional offices. One-year (12 issues) subscription rate U.S.A. and possessions \$21.95, Canada/Mexico \$31.95, all other countries \$49.95. All subscription orders payable in U.S. funds only, via international postal money order or check drawn on U.S. bank. Direct subscription orders and subscription related questions to Circuit Cellar INK Subscriptions, P. O. Box 698, Holmes, PA 19043.9613. Or call (800) 269.8301.

POSTMASTER: Please send address changes to Circuit Cellar INK, Circulation Dept., P.O. Box 698, Holmes, PA 19043.9613.

Cover photography by Barbara Swenson
PRINTED IN THE UNITED STATES

HAJAR ASSOCIATES NATIONAL ADVERTISING REPRESENTATIVES

NORTHEAST & MID-ATLANTIC
Barbara Best
(908) 741-7744
Fax: (908) 741-6823

SOUTHEAST
Christa Collins
(305) 966-3939
Fax: (305) 985-8457

WEST COAST
Barbara Jones
& Shelley Rainey
(714) 540-3554
Fax: (714) 540-7103

MIDWEST
Nanette Traetow
(708) 789-3080
Fax: (708) 789-3082

Circuit Cellar BBS—24 Hrs. 300/1200/2400/9600/14.4k bps, 8bits, no parity, 1 stop bit, (203) 871-1988; 2400/9600 bps Courier HST, (203) 871-0549

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

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

Entire contents copyright © 1995 by Circuit Cellar Incorporated. All rights reserved. Reproduction of this publication in whole or in part without written consent from Circuit Cellar Inc. is prohibited.

1 4 **Simulating Microprocessor Instructions in C**
by David Rees-Thomas

2 0 **No Emulator? Try a One-wire Debugger**
by Hank Wallace

2 4 **Using Spreadsheets to Simulate Digital Filters**
by Steven Kubis

2 8 **A RISC Designer's New Right ARM**
Writing Code for the ARM Processor
by Art Sobel

4 2 **Firmware Furnace**
Journey to the Protected Land: Serious CISC Meets the Taskettes
Ed Nisley

5 0 **From the Bench**
Getting By With Next to Nothing
Micro-power Wake-up Control
Jeff Bachiochi

5 7  **SPECIAL SECTION:**
Home Automation & Building Control

1 0 0 **Silicon Update**
UFO Alert!
Tom Cantrell

1 0 8 **Embedded Techniques**
Micros Behind Bars
John Dybowski

2 **Editor's INK**
Ken Davidson
It's Only Skin Deep

6 **Reader's INK**
Letters to the Editor

8 **New Product News**
edited by Harv Weiner

INSIDE ISSUE 54

115
ConnecTime
Excerpts from
the Circuit Cellar BBS
conducted by
Ken Davidson

128
Steve's Own INK
Steve Ciarcia
Hat Dance

97
Advertiser's Index

READER'S INK

THE TROUBLE WITH SUPERCAPS

I have never seen a formula estimating how long a supercap can back up NVRAM-which is surprising given the amount of interest in this subject lately. If you are going to design a supercap-backup system, you need to know how long it can last. If you've decided to use supercaps in your design, perhaps the following analysis will change your mind.

Let's compare the backup time of a 0.46-F supercap with that of the CR1632 lithium battery rated at 120 mAh (Dallas Semiconductor uses this battery in the popular NVRAM modules). The NVRAM used here will be typical of that used for battery-backed applications.

If the data retention current of the RAM is 1 μ A at 25°C and 12 μ A at 70°C. The battery can thus sustain the RAM contents at 25°C for:

$$\frac{120 \times 10^{-3} \text{Ah}}{1 \times 10^{-6} \text{A}} = 120,000 \text{ h or } 13.7 \text{ yr}$$

At 70°C, the backup time drops to:

$$\frac{120 \times 10^{-6} \text{Ah}}{12 \times 10^{-6} \text{A}} = 10,000 \text{ h or } 1.4 \text{ yr}$$

A different approach must be taken with a capacitor which uses a physical, rather than a chemical, process. The trick is to view this problem on the atomic scale. Since an ampere equals the flow of one coulomb per second and a coulomb equals 6.24×10^{18} e (electrons), we know that an amp-hour amounts to a charge of $(6.24 \times 10^{18} \text{ e/s}) \times 1 \text{ h} \times 3600 \text{ s/h}$ or 2.25×10^{22} e.

The formula $Q = CV$ relates the charge of a capacitor in coulombs to its voltage. If the capacitor is charged to 4 V, it will hold 0.47 F \times 4 V or 1.88 C of charge. If the capacitor is allowed to discharge to 2 V, it will then hold 0.47 F \times 2 V or 0.94 C of charge (the voltage range of 4-2 V is appropriate for the DS1210 nonvolatile controller IC). During this 2-V drop, the capacitor is allowed to source 1.88 C - 0.94 C = 0.94 C, which is atomically equivalent to 5.86×10^{18} e.

To put things back on familiar ground, let's convert this into a milliamp-hour rating:

$$1 \text{mAh} = \frac{(6.24 \times 10^{18} \text{ e/s}) \times 1 \text{ h} \times 3600 \text{ s/h}}{1000} = 2.25 \times 10^{19} \text{ e}$$

Therefore, the capacitor supplies:

$$\frac{5.86 \times 10^{18} \text{ e}}{2.25 \times 10^{19} \text{ e/mAh}} = 0.26 \text{ mAh}$$

At 25°C, the supercap provides a backup time of:

$$t = \frac{0.26 \times 10^{-3} \text{Ah}}{1 \times 10^{-6} \text{A}} = 260 \text{ h or } 10.8 \text{ days}$$

At 70°C, the backup time drops to 21.67 h! Yikes! What's gonna happen with the automated plant when your controller goes down on a summer weekend?

Obviously, the decision to use supercaps for critical system backup must be made with care. I like to use a supercap as an emergency backup for the backup battery—a 3.3-F, 2.5-V cap is a good choice which supplies a couple of hundred hours of secondary backup.

The following table should help put the relative capacities of batteries and supercaps in perspective:

Power Source	Capacity		Backup Time		
			@25°C	@40°C	@70°C
CR1632	144F	120mAh	13.7 yrs.	5.7 yrs.	1.1 yrs.
supercap	0.47 F	0.26 mAh	10.8 days	4.5 days	21.6 h

Note the very large equivalent capacitance of the batteries and the small equivalent capacity of capacitors.

Dale Nassar
Amite, LA

Contacting Circuit Cellar

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

Mail: Letters to the Editor may be sent to: Editor, The Computer Applications Journal, 4 Park St., Vernon, CT 06066.

Phone: Direct all subscription inquiries to (800) 269-6301. Contact our editorial offices at (203) 875-2199.

Fax: All faxes may be sent to (203) 872-2204.

BBS: All of our editors and regular authors frequent the Circuit Cellar BBS and are available to answer questions. Call (203) 871-1988 with your modem (300-1 4.4k bps, 8N1).

Internet: Electronic mail may also be sent to our editors and regular authors via the Internet. To determine a particular person's Internet address, use their name as it appears in the masthead or by-line, insert a period between their first and last names, and append "@circellar.com" to the end. For example, to send Internet E-mail to Jeff Bachiochi, address it to jeff.bachiochi@circellar.com. For more information, send E-mail to info@circellar.com.

NEW PRODUCT NEWS

Edited by Harv Weiner

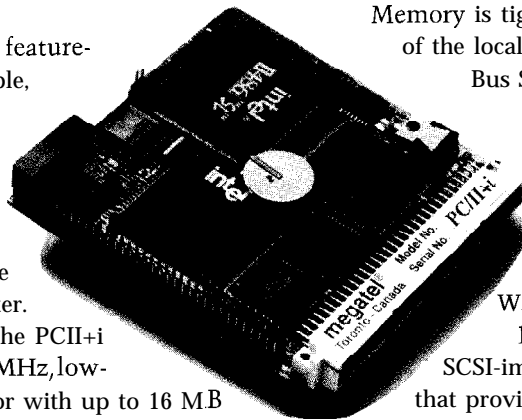
486 EMBEDDED PC

Megatel has released a feature-packed 80486 PC-compatible, single-board computer.

The PC/II+i is packaged on a 100 x 100 mm board and is available in either a PC/104 or ISA bus-compatible format with the addition of Megatel's adapter.

Features available on the PCII+i include either a 25- or 33-MHz, low-power, Intel 80486 processor with up to 16 MB of user DRAM, 256 KB of BIOS flash EPROM, AT-compatible BIOS, 2 MB of flash disk, a full 32-bit DRAM data bus, and 8 KB of built-in cache with floating-point units. Also on the board are a SCSI host adapter, floppy-disk controller, S-VGA video and LCD controller, and Ethernet interface. Standard I/O features include two IBM-compatible RS-232 serial ports and one BIOS-compatible RS-232 serial port, a general-purpose parallel I/O port, real-time clock with battery backup, and the 16-bit ISA I/O bus. CMOS technology is used to reduce power consumption to approximately 6 W and +5 V only.

Performance of the PC/II+i is increased by incorporating the capacity for 16 MB of on-board memory.



Memory is tightly coupled, thus enhancing the operation of the local cache. Chips & Technology's 65530 Local

Bus S-VGA controller with up to 1 MB of video RAM facilitates many of the higher-resolution video modes. A complete legal BIOS (in flash EPROM) that boots standard versions of PC, MS, or Novell DOS is provided. The PC/II+i runs popular PC software packages including Windows 3.1.

Full SCSI host adapter support includes a SCSI-implemented AT hard-disk-drive controller that provides up to 50% increase in hard-disk performance over IDE. DOS and low-level formatting are accomplished by a single program. Other SCSI features include a full ASP1 shell interface including drivers for popular CD-ROMs, magneto-optical drives, and so on. Also included is SCSI-extension software, which offers automatic adjustment of AUTOEX EC. BAT and C O N F I G . S Y S files, spanning capability, and a single-install menu.

The PC/II+i sells for \$895 in quantity.

Megatel Computer Corp.

125 Wendell Ave. • Weston, ON • Canada M9N3K9
(416) 245-2953 • Fax: (416) 245-6505

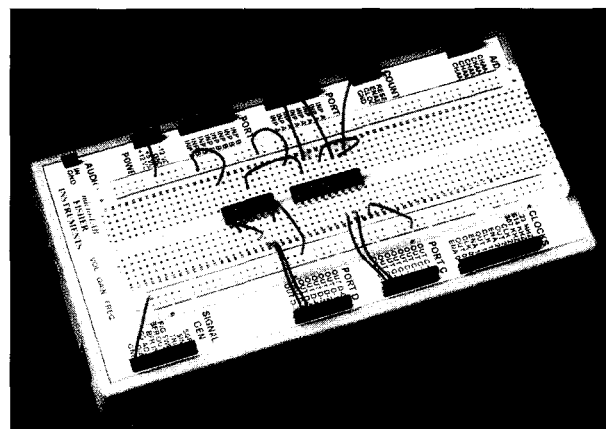
#500

UNIVERSAL COMPUTER INTERFACE

Fisher Instruments introduces a universal computer interface providing design aid for engineers, experimenters, and students. **micro-LAB** functions with virtually any computer using an RS-232 serial interface at 300-19,200 bps with no handshaking required. micro-LAB enables a PC to power and control designs in any programming language.

The micro-LAB package includes a solderless breadboard with a function generator that produces sine, square, and triangular waveforms with a sweep input. The unit features three crystal-controlled clock-frequency sources, three 16-bit programmable counters, and one 8-bit event counter. Three A/D channels dedicated to DC measurements, one A/D channel devoted to AC measurements, and one 8-bit D/A converter are on board. Two independent 8-bit TTL-compatible input ports, two independent 8-bit output ports, and a 300-mW audio amplifier with internal speaker are also included. The unit measures 7.5" x 3.5" x 1.5".

micro-LAB sells for \$249.95 and includes sample application programs, sample graphics drivers, RS-232 interconnecting cable, and a user's manual. The Power Pack (ELPAC WM-1 13TT) option is an additional \$49.95 and a demo disk outlining micro-LAB's capabilities is available for \$5.00.



Fisher Instruments

20611 E. Bothell-Everett Hwy., Ste. 232 • Bothell, WA 98012 • (206) 489-9153 • Fax: (206) 487-1528

#501

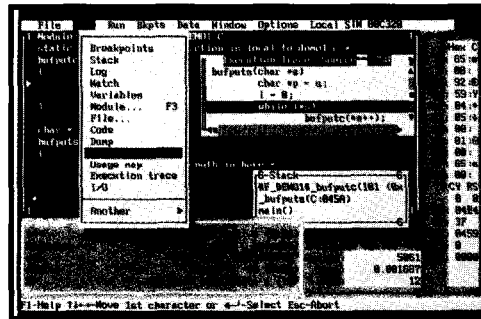
NEW PRODUCT NEWS

8051 DEBUGGER

ChipTools has released Version 3.1 of **ChipView-51**, a high-level debugger for 8051 C compilers. It is key compatible with Borland's Turbo Debugger, and is available in three versions: a high-performance simulator and debugger, a high-level user interface for Nohau's EMUL5 1-PC, and a ROM-monitor debugger.

With ChipView, the Turbo C programmer can instantly debug code in the embedded-systems environment. ChipView presents over 14 different views of the user's program, including all of Turbo Debugger's views. It can display a C-level call stack, which shows nested function calls along with their arguments.

The ChipView- simulator provides full support for Dallas Semiconductor's DS5000, DS5001, and high-speed 80C320. The user program can interact with real on-chip and off-chip I/O, such as A/D converters, timers, ports, or custom memory-mapped I/O. Remote I/O via the PC's COM ports lets the user attach real serial I/O



to the simulator. A quick I/O window simulates a display terminal for interactive I/O even while the simulation engine is running.

The ChipView ROM monitor also features the quick I/O window. The host-target serial I/O link automatically time shares, permitting the user program to also use the serial port for I/O to the display window or another serial device.

The ChipView- 1 Nohau emulator version provides support for every production board and pod from Nohau.

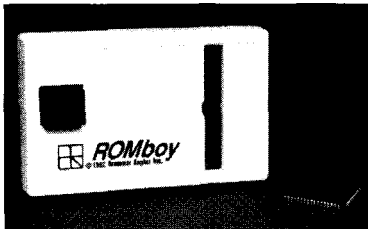
The ChipView- 1 Version 3.1 simulator sells for \$795, the emulator version for \$595, and the ROM-monitor version for \$795. A combo package is available for \$995. System requirements include an IBM AT or compatible with 3 MB of RAM and a hard disk.

ChipTools

1232 Stavebank Rd. • Mississauga, ON Canada L5G2V2
(905) 274-6244 • Fax: (905) 891-2715

#502

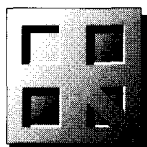
The BEST in ROM emulation technology:



- 1 Mbit
- 100ns
- Price \$295

ROMboy includes a 10 day, no-risk money back guarantee!

Call Today — 800-776-6423



**Grammar
Engine
Inc.**

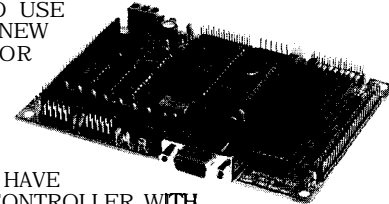
921 Eastwind Dr., Suite 122
Westerville, OH 4308 1
614/899-7878
Fax 614/899-7888

Now Available
4 Mbit

ROMBOY!

SINGLE BOARD COMPUTER

THAT'S RIGHT! \$129.95 FOR A FULL FEATURED SINGLE BOARD COMPUTER FROM THE COMPANY THAT'S BEEN BUILDING SBC'S SINCE 1985. THIS BOARD COMES READY TO USE FEATURING THE NEW 30535 PROCESSOR WHICH IS 8051 CODE COMPATIBLE. ADD A KEYPAD AND AN LCD DISPLAY AND YOU HAVE A STAND ALONE CONTROLLER WITH ANALOG AND DIGITAL I/O. OTHER FEATURES INCLUDE:



- UP TO 24 PROGRAMMABLE DIGITAL I/O LINES
- 8 CHANNELS OF FAST 8/ 10 BIT A/D
- UP TO 4, 16 BIT TIMER/COUNTERS WITH PWM
- UP TO 3 RS232/485 SERIAL PORTS
- BACKLIT CAPABLE LCD INTERFACE
- OPTIONAL 20 KEY KEYPAD & INTERFACE
- 160K OF MEMORY SPACE, 64K INCLUDED
- 8051 ASSEMBLER & ROM MONITOR INCLUDED

EMAC, inc.

618-529-4525 Fax 4570110 BBS 529-5708
P.O. BOX 2042, CARBONDALE, IL 62602

NEW PRODUCT NEWS

IN-CIRCUIT EMULATOR

The Signum in-circuit emulator offers real-time, transparent emulation for the entire Intel 80C 186 family of microcontrollers, including the XL, EA, EB, and EC versions. The USP-186 eases the development of the software and hardware of embedded-controller products in telecommunications, image processing, modems, robotics, and other high-speed applications.

The USP-186 connects to any IBM 386/486-compatible host computer via a serial port and users download and upload programs at 115 kbps.

The emulator emulates at speeds up to 26 MHz and comes equipped with 1 MB of overlay memory, which may be enabled in 256-byte blocks. The trace-buffer memory is 32,768 entries deep by 80 bits wide, has filtering controls, and includes a real-time stamp with a 100-ns resolution.

With the USP-186, a user can debug a real-time application without stopping the processor. With the aid of dual-ported memory, the user can view and modify program and data memory, define breakpoints, and enable the trace buffer while the processor is running.

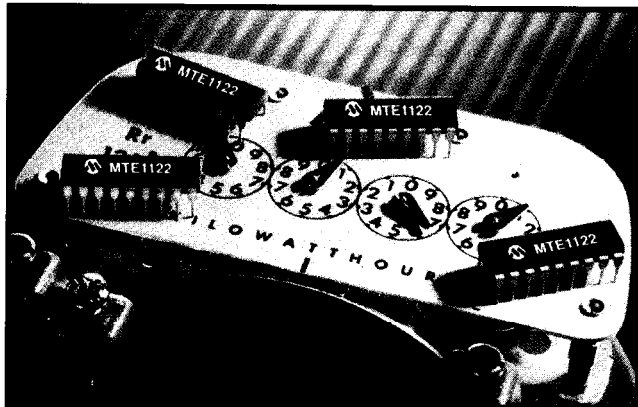
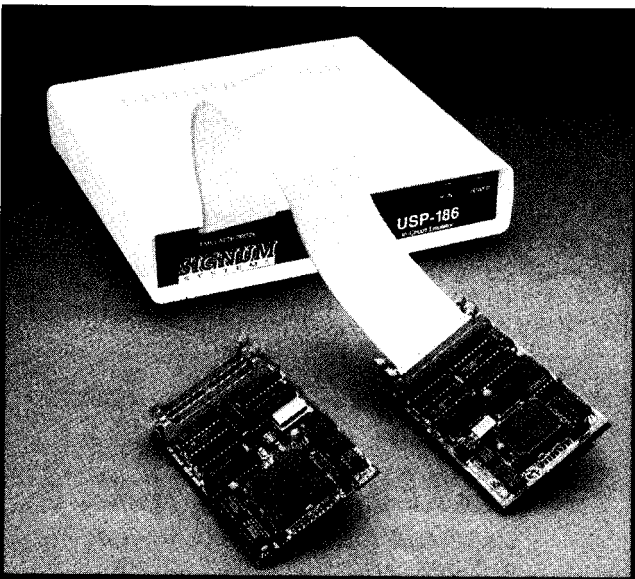
A special windowing interface with a mouse makes the user interface fast and simple to use. An integrated source-level debugger for C provides source-code line stepping, local variable display, and support for all variable types including nested structures and arrays.

The Signum USP-186 In-Circuit Emulator is priced from \$7890 (20 MHz) to \$8290 (26 MHz).

Signum Systems

171 E. Thousand Oaks Blvd., Ste. 202
Thousand Oaks, CA 91360
(805) 371-4608 • Fax: (805) 371-4610

#503



ENERGY MANAGEMENT CONTROLLER

Microchip introduces a device which reduces total energy consumption by up to 30% or more in a wide variety of electrical product applications. Typical applications for the MTE1122 Energy Management Controller encompass all residential, commercial, and industrial equipment which use fractional-horsepower AC motors. Potential applications include water-filtration systems, sump pumps, refrigerators, cooling fans, compressors, and air-conditioning units.

The MTE1122 integrates Microchip's 8-bit RISC-based PIC16/17 microcontroller technology with proprietary power-management firmware to allow AC-induction-motor applications to be more energy efficient. This saves energy costs by reducing utility demand. The energy consumed by the motor more closely matches its work.

The Energy Management Controller operates by digitally monitoring the

motor load and then controlling power consumption thousands of times per second. Most AC induction motors require large currents under light or even no-load situations. The unique algorithm in the MTE1122 monitors the AC signal and senses when the motor is consuming more power than is required. The device then modifies the AC signal so the motor can continue its rotational speed with less power.

The MTE1122 is available in 18-pin PDIP and SOIC packages and features 5-V operation and automatic power-on reset. List price for the MTE 1122 Controller (PDIP version) is \$7.49 in 1000-piece quantities.

Microchip Technology, Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
(602) 786-7200
Fax: (602) 899-9210

#504

NEW PRODUCT NEWS

'386EX EMULATOR

Softaid has released an in-circuit emulator for Intel's '386EX embedded processor. The **UEM-386EX** offers a high-performance '386 development environment operating under Windows at speeds of 25 MHz.

Real-time trace is included, overcoming a shortcoming found in many low-priced tools. Trace is essential for debugging interrupt and DMA-based code since stopping the program at a breakpoint invariably corrupts the integrity of the emulation. The UEM-386EX includes a 4-KB trace buffer, generating views of the data as raw machine instructions, C source, or intermixed C and disassembled code. Triggers qualify trace-data collection, limiting acquisition to events of interest—all in real time.

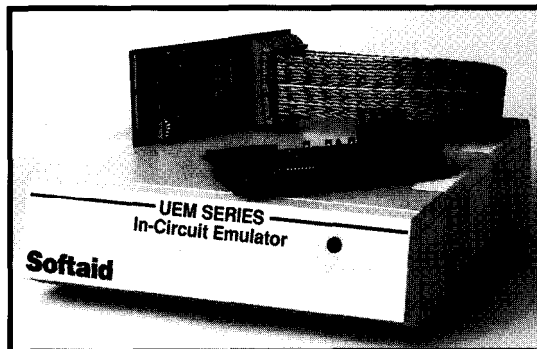
The UEM-386EX comes standard with an integral performance analyzer that monitors the time spent in up to **255** routines simultaneously, maintaining accuracy better

than 100 ns. The performance analyzer quickly finds software bottlenecks.

Softaid's development tools come with the UEM-386EX emulator and SLD for Windows. The emulator gives firmware engineers the raw resources needed to debug an embedded system, while SLD provides a shell to debug C and assembly language code. All compilers are supported.

The UEM-386EX offers an upgrade path for developers switching from older '186 designs to the '386EX. A simple pod swap lets the UEM work with any version of the '186 and the '386EX. The emulator covers the entire embedded x86 family.

The UEM-386EX emulator with SLD for Windows costs \$9000.



Softaid, Inc.
8310 Guilford Rd.
Columbia, MD 21046
(410) 290-7760
Fax: (410) 381-3253

#505

MOVE OVER INTEL MICROMINT SOURCES 80C52 CMOS BASIC CHIP

Micromint has a more efficient software-compatible successor to the power-hungry Intel 8052AH-BASIC chip. The 80C52-BASIC chip was designed for industrial use and operates beyond the limits of standard commercial-grade chips. Micromint's 80C52-BASIC chip is guaranteed to operate flawlessly at DC to 12 MHz over the entire industrial temperature range (-40°C to +85°C). Available in 40-pin DIP or PLCC

80C52-BASIC chip \$19.00
OEM 100-Qty. Price \$12.00
BASIC-52 Prog. manual \$15.00

MICROMINT, INC.
4 PARK ST. • VERNON, CT 06066
TO ORDER CALL
1-800-635-3355

C COMPILER FOR PIC CONTROLLERS

- Integrated software development environment including an editor with interactive error detection/correction.
- Access to all hardware features from C.
- Includes libraries for RS232 serial I/O and precision delays.
- Efficient function invocation mechanism allowing call trees deeper than the hardware stack.
- Special built-in features such as bit variables optimized to take advantage of unique hardware capabilities.
- Interrupt and A/D built-in functions for the C71.
- Easy to use high level constructs:

```
#include <PIC16C56.h>
#use Delay(Clock=2000000)
#use RS232(Baud=9600,Xmit=pin_1,RCV=pin_2)

main () {
    printf("Press any key to begin\n");
    getc();
    printf("1 khz signal activated\n");
    while (TRUE) {
        outgit_high(pin_8);
        delay_us(500);
        output_low(pin_8);
        delay_us(500);
    }
}
```

PCB compiler \$99 (all 5x chips)
PCM compiler \$99 ('64, '71, '84 chips)

Pre-paid shipping \$5
COD shipping \$10

CCS, PO Box 11191, Milwaukee WI 53211
414-781-2794 x30

NEW PRODUCT NEWS

INTERMITTENT TESTING BY POWER-ON CYCLING

Power-on and intermittent failures can be easily diagnosed with a new piece of test equipment from MicroTools. **Poe-it** is intended to provide a one-step solution to the problem of power-on testing.

Intermittent problems, caused by hardware and software, often occur after a power up. The problems are difficult to duplicate, and fixes are sometimes questionable. Some intermittent problems include improper hardware initialization, temperature-sensitive race conditions, vibration-sensitive interconnects, noisy or noise-susceptible power circuitry, unprotected interrupt windows, and power-on system-test problems. Systems may need to be tested for thousands of cycles before such problems appear.

Pot-it can be used early in the design cycle to uncover such problems. Pot-it is designed around an 8051 family part and features two high-speed input counters with 5-VDC inputs, one optically isolated 10–30-VDC input, one 120-VAC @ 10-A cycled output, and one 5-A relay contact. Its user interface consists of a 1-line, 16-character LCD display, and 4-button keypad. A simple, menu-driven interface sets on and off times of each output with a 10-ms resolution, resets input counters, starts and stops the test, and lets cycle counters for all inputs and outputs be viewed.

Pot-it sells for \$295.



MicroTools, Inc.

P.O. Box 624 • 714 Hopmeadow St., Ste. 14 • Simsbury, CT 06070 • (800) 651-6170 • Fax: (203) 651-0019

#506

CD-ROM ACCELERATOR

A CD-ROM Accelerator, which makes CD-ROM applications perform as fast as if they were running from a hard drive, has been announced by Ballard Synergy Corp. **d-Time¹⁰ V1.1** sets a new "ease of use" milestone for a CD-ROM accelerator with a Windows help program that has full-motion video.

When a quad-speed CD-ROM is accelerated, access times improve by 20 times (from 200 ms to 10 ms) and data transfer rates by about 8 times. Slower CD-ROM drives see even more dramatic

improvements. Unlike RAM caches for CD-ROM, **d-Time¹⁰** copies the critical data from CD-ROM to the hard disk. **d-Time¹⁰** removes all glitches and pauses from multimedia sequences and saves a fifth of a second for each CD-ROM access. Twenty-minute, CD-ROM database searches are reduced to one minute.

d-Time¹⁰ uses state-of-the-art, patent-pending technology to make the CD-ROM perform as fast as the disk drive. As the CD-ROM is used, **d-Time¹⁰** automatically updates the acceleration file on the hard disk with the contents of the CD-ROM disks. Even if a power failure occurs, all information is retained

since it is on the hard drive. **d-Time¹⁰** can create a time log containing the CD-ROM sector IDs, which can be used to re-create the exact contents of the acceleration file even on a different machine. Time logs for over 60 titles are included on the **d-Time¹⁰** CD-ROM. By using the time log for a particular CD-ROM, the slow access of the CD-ROM can be avoided even for a first-use application.

d-Time¹⁰ features a quick install, which uses standard default values and includes extensive on-line help. It supports enhanced IDE hard drives, Novell DOS7, and 4DOS. The **d-Time¹⁰** CD-ROM is required only during installation.

d-Time¹⁰ is a pure software solution, so there are no switches to set or hardware to plug in. All that is necessary is to decide how much disk space to give the accelerator file.

d-Time¹⁰ sells for \$69.95.

Ballard Synergy Corp.
10715 Silverdale Way,
Ste. 208
Silverdale, WA 98383
(206) 656-8070
Fax: (206) 656-8205

#507

NEW PRODUCT NEWS

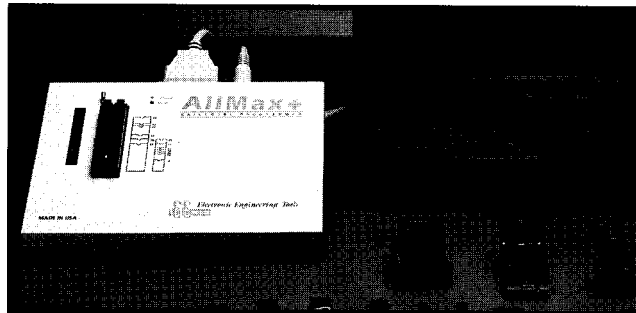
UNIVERSAL DEVICE PROGRAMMER

Electronic Engineering Tools has announced a new Universal Device Programmer, which connects to a PC through a parallel port or a high-speed, parallel-interface card simply by using a switch. AllMax+ is a software-expandable programmer that supports a wide variety of programmable devices as well as testing digital ICs, SRAM, and DRAM.

AllMax+ interfaces with IBM-compatible personal computers. The operating software features a user-friendly interface that includes

pull-down menus, a macro facility for batch-file execution, and virtual memory management to deal with very large files. AllMax+ software reads output from most compilers in JEDEC formats such as CUPL, PALASM, OPAL, and ABEL. It also includes test-vector capability, multiarray fuse-map editor, DOS shell-handling utilities, and file-format handler.

Devices supported are PLDs including AMD Mach family; bipolar PROMs; EEPROMs up to 16 Mb; microcontrollers such as the Microchip PIC series, Motorola MC68000, and Zilog Z86 series; and serial EEPROMs. The AllMax+



hardware, including the standard 48-pin ZIF socket, minimizes additional adapter usage for regular DIP-type devices.

The AllMax+ package sells for \$745.00. It includes a 48-gold-pin ZIF-socket programming module, universal-switching power supply (100-250 VAC), 6' printer cable, installation

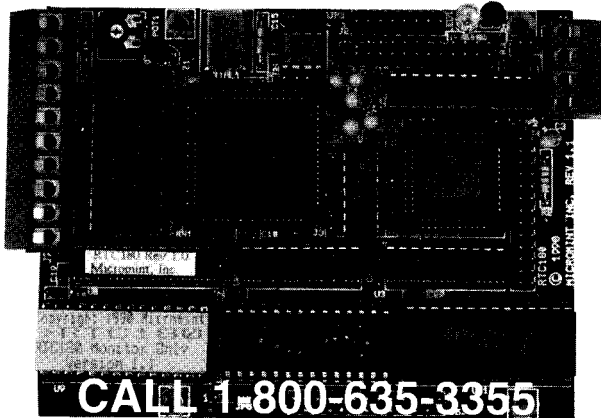
disk, and manual. Other programming modules and sockets are available.

Electronic Engineering Tools, Inc.
544 Weddell Dr., Ste. 6
Sunnyvale, CA 94089
(408) 734-8184
Fax: (408) 734-8185

#508

STOP LOOK LISTEN

Odds are that some time during the day you will stop for a traffic signal, look at a message display or listen to a recorded announcement controlled by a Micromint RTC180. We've shipped thousands of RTC180s to OEMs. Check out why they chose the RTC180 by calling us for a data sheet and price list now.



CALL 1-800-635-3355

MICROMINT, INC.

4 Park Street, Vernon, CT 06066
(203) 871-6170 • Fax (203) 872-2204



in Europe: (44) 0285-658122 • in Canada: (514) 336-9426 • in Australia: (3) 467-7194 • Distributor Inquiries Welcome

FEATURES

14 Simulating
Microprocessor
Instructions in C

20 No Emulator? Try a
One-wire Debugger

24 Using Spreadsheets to
Simulate Digital Filters

28 A RISC Designer's
New Right ARM

FEATURE ARTICLE

David Rees-Thomas

Simulating Microprocessor Instructions in C

Although burning EPROMs no longer costs a bundle, it still takes time and isn't very efficient for debugging. To cut labor costs and better learn the micro he's using, David creates his own simulator.



When you do embedded controller projects on a limited budget, you need every bit of inexpensive debugging help you can get. EPROM and EEPROM versions of your favorite processor are great because they let you test your code, fix it, and try again. But, the burn-and-pray method of debugging is inefficient at best and downright frustrating much of the time.

For example, suppose your project has to use BCD (binary-coded decimal) arithmetic. Your CPU doesn't have a DAA (decimal-adjust accumulator) instruction, so you write a subroutine to do the equivalent function. Can you readily test all possible combinations of input to that routine? If it doesn't work 100% the first time, how many patches will it take to make it work? How many E(E)PROMs will you have to burn to be sure?

Here's where a software simulator can save hours of development time. Running on a PC or Mac, the simulator lets you step through your code line by line, manipulate registers, watch changes in memory, and monitor the CPU's flag bits, all with a few keystrokes. Need a **B EQ** (branch if equal) instead of a **B NE** (branch if not equal) at address `$F0C3` (that's `0F0C3h`)

Listing 1—The target microprocessor's instruction set is represented by an array of structures containing details about each instruction. Many high-level languages, including C, make building such an array very intuitive and are well suited for use in developing a simulator.

```
a)
struct instruction {
    char mnemonic[8];      /* opcode mnemonic in ASCII */
    int opcode;            /* opcode in binary */
    int n-bytes;           /* length in bytes */
    int n-cycles;         /* machine (E) cycles */
    int mode;              /* code for addressing mode */
    void(*fcn)();         /* ptr to implementation fcn */
};

b)
struct instruction instruct[] =

    "lda ", 0xA6, 2, 2, 4, lda /* implements LDA immediate*/
    :
    :
};
```

for you nonMotorolans)? Simply change the contents of that location from \$26 to \$27, and run your simulation again. You can clean out a whole handful of bugs like this in the time it takes to erase one EPROM.

OK, that's nice, but where can you get a simulator cheap? There are lots of good simulators out there if you can afford them, but what can you do on a limited budget? With a mainstream controller like the 68HC11, you often can find a freeware or shareware simulator by searching the bulletin boards.

On the other hand, there may not be any simulator available for the obscure Nominal Macro XYZ223 chip in your latest project. If time is no object, you might want to try writing your own.

This article describes the approach I used to develop a simulator for the Motorola 6805 family of microcontrollers.

SIMULATOR BASICS

In its simplest form, a simulator lets you execute the functions typically found in the ROM-monitor firmware on an evaluation board. You can:

- examine and change memory contents and CPU registers
- load a program into memory

- disassemble machine code in memory
- execute machine instructions in memory continuously or step by step
- set break and watch points to monitor program execution

You can implement the first three simulator functions fairly easily in any high-level language. An array of **bytes**(**unsigned char**inC)can represent the processor's memory or I/O address space. Bytes and words (**unsigned int**) can be used for 8- and 16-bit CPU registers. Loading a program is usually a fairly simple matter of translating the S19 or Intel hex file output by an assembler from ASCII characters to binary and saving the results in the correct elements of the memory array.

READING INSTRUCTIONS

Disassembly or regeneration of the original assembler source code from machine instructions is a somewhat larger task. Each machine instruction can be represented as a unique binary or hexadecimal number stored in memory. We can use that number as an index into some sort of table and then print out the corresponding assembler mnemonic and operand. The problem is that it's hard to tell what's an opcode and what's an

operand. For example, suppose the three memory locations starting at address \$0400 (400 hex) each contain the byte \$A6:

0400 A6 A6 A6...

One of those bytes is the opcode or machine code for a 6805 **LDA** (**load accumulator**) instruction—but which one?

As with all good questions, the answer is "it depends." If the CPU has just been reset, and the reset vector contains \$0400, then the first A6 is the opcode. The same applies if the CPU has just completed execution of a previous instruction. In both cases, the CPU's program counter (PC) contains \$0400, and the processor is ready to fetch an opcode. The CPU then reads the contents of \$0400 and increments the program counter.

What's the next A6?

Once again, it depends. Since the processor has just completed an opcode fetch, the meaning of the next byte depends on how this machine instruction is decoded by the CPU. In the 6805 family, the load accumulator can be represented by six different opcodes: A6, B6, C6, D6, E6, and F6. Each instruction puts one byte of data into the accumulator. Where that byte comes from (i.e., the effective address of the byte) depends on which of the six opcodes the CPU fetched.

In decoding a machine instruction, the processor determines two things: the actual operation to be performed (load, add, or compare) and the instruction's addressing mode. From the latter and, in some cases, the contents of a CPU register, the processor computes an effective address. In our example, A6 represents a load accumulator in the immediate mode. The operand—the actual data loaded into the accumulator—is in the location immediately following the opcode. The complete instruction is two bytes long. The third A6 then becomes the opcode of another LDA instruction.

Getting back to our disassembler, we can see that the table entry for opcode number 166 (\$A6) might contain the following items:

- an assembler mnemonic string in ASCII characters (**LDA**)
- the length of the instruction in bytes (2)
- a code to indicate addressing mode (4-an arbitrary choice)

A table of 256 such entries covers the entire 6805 opcode map including illegal opcodes, which are values with no corresponding machine instruction. Members of the 6805 family share a single-page opcode map (i.e., every opcode occupies a single byte). Other processors such as the 68HC11 or Z80 have a number of two-byte opcodes, but the number of different values of the extra byte or prebyte usually is quite small.

Now, with a little bit of extra effort, we can disassemble the sequence A6 A6 and print:

```
0400 A6 A6 LDA #A6
```

If we encounter an illegal opcode we can print any suitable indicator, such as **I LLOP** or just *******:

```
0416 41 ***
```

EXECUTING INSTRUCTIONS

One thing a disassembler can't do is tell the difference between code and data. \$41 is not a legal opcode for a 6805, but it is the ASCII value for an "A". A disassembler can identify a complete instruction such as J M P \$0420. It can't follow program flow, so it doesn't know enough to jump over the character string "ABORT" starting at \$0416, say, and pick up again at \$0420. What we need is a way to execute each instruction in turn, so we can follow the program flow.

Simulating the execution of a microprocessor instruction is not all that difficult once you've built the instruction table. We can break the execution into a sequence of seven steps:

1. Fetch the opcode from the memory location defined by the contents of the simulated program counter
2. Increment the PC
3. Determine the instruction's addressing mode

Listing 2-a) The `lda()` function copies one byte from simulated memory to the simulated-accumulator register, setting flag bits if the value of the byte is zero or negative. The `where()` function (see Listing 3) determines the location in memory which contains the original data. b) The flag bits (condition codes in Motorola) are implemented as 1-bit fields in the structure CC. Flag bits generally reflect the result of the most recently executed instruction(s). The `I` bit, a Motorola exception, is cleared or set by specific instructions to enable or disable CPU interrupts.

```
a)
void lda(byte opcode) {          /* Load Accumulator      */
    word ea;                     /* Effective Address      */

    ea = where(opcode);         /* compute Effective Address */
    A = memory[ea];            /* load the accumulator from */
    /* location addressed      */
    CC.N = (A & 0x80) ? 1 : 0; /* set sign flag if MSB = 1 */
    CC.Z = (A) ? 0 : 1;        /* set zero flag if data = 0 */
}

b)
struct ccr {                    /* Condition Code Register */
    unsigned int C: 1;         /* carry flag              */
    unsigned int Z: 1;         /* zero flag               */
    unsigned int N: 1;         /* negative (sign)        */
    unsigned int I: 1;         /* interrupt mask          */
    unsigned int H: 1;         /* half-carry flag        */
} CC;
```

4. Read the operand(s) if any and compute an effective address (EA)
5. Increment the PC as required so that it points to the next instruction
6. Modify any registers and/or memory locations that are affected by the instruction
7. Set or clear any condition code (flag) bits that are affected by the instruction (carry, sign, zero, etc.)

An instruction table looks after the first five steps. The last two require you to write a set of what I call *implementation functions*. Each implementation function performs a machine instruction by manipulating the contents of the simulated registers, memory, and condition codes. We could write a separate function for each opcode, but it's simpler to lump all of the variations of one instruction, such as **LDA**, into a single routine.

I chose to implement my simulator in C partly as a learning exercise and partly because of some useful features of the language. I've been referring to an instruction table as a basic part of the program. As you can see in Listing 1a, a single entry in this table is a structure of type `instruction`. The entire instruction table or

opcode map is represented by an array of 256 instructions (see Figure 1b).

I declared the CPU registers A (accumulator), X (index register), PC (program counter), and SP (stack pointer) as global variables (`unsigned char` or `int` as appropriate).

You may have noticed some additions to the original table entry. The variable `opcode` just repeats the position of a specific entry in the array, but it adds readability and makes life a bit easier later on. The number of machine cycles that it takes to execute an instruction is tracked with `n_cycles` (E cycles in Motorola). The pointer to the specific C function, which actually implements the instruction `opcode`, is the most important. We'll look at an example of an implementation function shortly.

The first two steps in the execution of a microcontroller instruction include fetching the opcode and incrementing the program counter. We can do that in one line of C:

```
opcode = memory[PC++];
```

The value in the simulated program counter PC (an `unsigned int`) identifies the location of the next

instruction to execute. The contents of that location (i.e., the opcode) then becomes an index into the instruction array. Each member of that array is a C structure. So, for example, we can refer to the addressing mode that corresponds to a given opcode as `instruction.Copcode1.mode`. Better yet, `allit` takes to execute an instruction is one line:

```
instruct[opcode].fcn();
```

The work comes in writing the implementation functions for each of the microcontroller's instructions. It's not difficult, but you have to keep track of the details. Going back to my earlier example, opcode A6 in a Motorola 6805 leads us to an instruction array member that looks like this:

```
"LDA", 0xA6, 2, 2, 4, Ida
```

This instruction loads the accumulator with the contents of the memory location immediately following the opcode. The hex value of the opcode is A6. The instruction is 2 bytes long, and it takes 2 clock cycles to execute. Code 4, by my convention, indicates the immediate addressing mode. 1 da is the name of the C function performing the simulated load-accumulator instructions. The extra spaces following the mnemonic LDA help to format the output of the disassembler. Listing 2a shows the 1 da implementation function.

In the listing, I define `byte` and `word` as `datatype of unsigned char` (8-bit) and `unsigned int` (16-bit), respectively. The function `where` (discussed in more detail later) computes the effective address `ea`, which in this case is the memory location containing the data to be loaded into

the accumulator. The next line does the actual loading of A.

WHAT ABOUT THE CPU FLAGS?

Motorola microcontroller instructions are much more likely to have an effect on the flags or condition codes than those of a Z80. Thus, every 6805 LDA instruction affects both the Z(ero) and N(egative) flag bits according to the value loaded.

I represented the condition-code register as another structure—in this case, a bit-field named CC (Listing 2b). The last two lines of the `lda` function implement this manipulation of the flags:

```
if bit 7 of A is set,
    then set the N flag,
    else clear it;
if A equals 0 after the load,
    then set the Z flag,
    else clear it.
```

COMPUTING THE EFFECTIVE ADDRESS

The `where` function used to compute effective address is common to most of the implementation functions. The exceptions implement instructions, such as COMA or CLRX, which use the inherent addressing mode. These instructions don't require a memory access other than the opcode fetch. The current version of `where` (Listing 3) takes advantage of the fact that the addressing mode is encoded in the most-significant four bits of a 6805 opcode. All immediate-mode instructions, for example, not just LDA, have hex values starting with \$A.

The remaining LDA opcodes—\$B6, \$C6, \$D6, \$E6, and \$F6—cover direct, extended, and three varieties of indexed addressing. Extended addressing is the most obvious. The complete effective address is contained in the two bytes following the opcode. Direct addressing is similar, except that only the least-significant byte of the effective address follows the opcode. The most-significant byte is always \$00. An indexed effective address is formed by adding the contents of the X register (8 bits wide) and an unsigned offset. The offset may be zero, one, or

Listing 3—The function `where()` computes the memory address of the source or destination in a data (load, store, test, or modify) instruction or the destination of an absolute jump or jump-to subroutine. For a conditional branch instruction, `where()` returns a signed offset to be added to the contents of the program counter if the condition is true. Specific bits in the instruction opcode determine the addressing mode (i.e., exactly how the address computation is to be performed).

```
word where(byte opcode) {           /* compute effective address*/
    word temp;

    switch (opcode>>4){             /* MS 4 bits is mode */
        case 0x00: case 0x01:       /* bit manipulation inst */
        case 0x03: case 0x0B:       /* direct addressing */
            return(memory[PC++]);    /* MS byte is always 00*/

        case 0x02:                  /* relative branches */
            return(sex(memory[PC++])); /* return offset, not EA */

        case 0x06: case 0x0E:       /* indexed. 8-bit offset */
            return(X+memory[PC++]);  /* follows op-code*/

        case 0x07: case 0x0F:       /* indexed, zero offset*/
            return(X);              /* is 1-byte instruction */

        case 0x0A:                  /* immediate mode*/
            return(PC++);           /* data follows op-code */

        case 0x0C:                  /* extended address*/
            temp = memory[PC++] << 8; /* MS byte follows opcode */
            return (temp + memory[PC++]); /* LS byte follows MS*/

        case 0x0D:                  /* indexed, 16-bit offset */
            temp = memory[PC++] << 8; /* is 2 bytes after opcode */
            return (X + temp + memory[PC++]);

        default:
            return (0xFFFF);
```

two bytes in length, depending on the opcode.

One addressing mode is treated a bit differently. Case 0x02 (relative branches) returns a signed offset rather than an effective address. It isn't quite as interesting as it looks, though. The **sex** function merely sign-extends an S-bit two's complement offset to 16 bits. The branch-instruction-implementation functions add the result to the current contents of the program counter to give the location of the next instruction.

CONCLUSION

This discussion should give you enough information to start writing your own microcontroller-simulation package. I haven't gone into details about the user interface since that's a matter of personal preference. My original version, written as a teaching tool at the British Columbia Institute of Technology, simply duplicated the ROM-monitor interface on the 146805E2 boards used in the lab. With a simulated on-chip timer and interrupts generated from the PC keyboard, the program helped several classes of BCIT students to unravel the mysteries of microcontroller-instruction execution.

A more recent revision simulating the MC68HC05J2 enabled me to find a bug in Motorola's original documentation of the half-carry flag. If nothing else, writing your own simulator gives you new insight into the operation of your favorite microcontroller. ☐

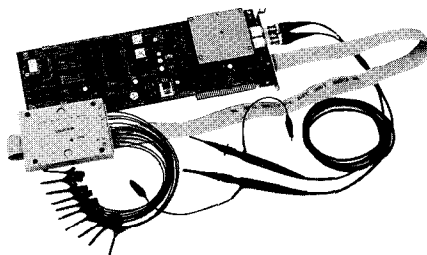
David Rees-Thomas has a B.Sc. in chemistry and math from Queen's University and a diploma in Electronics Technology from Northern College in Kirkland Lake, Ontario. For the last ten years, he has been teaching at the British Columbia Institute of Technology in Burnaby, BC, where he specializes in microcontrollers and data communications. David may be reached at resd2215@bcit.bc.ca.

IRS

- 401 Very Useful
- 402 Moderately Useful
- 403 Not Useful

PC-Based Instruments 200 MSa/s DIGITAL OSCILLOSCOPE

**HUGE BUFFER
FAST SAMPLING
SCOPE AND LOGIC ANALYZER
C LIBRARY W/SOURCE AVAILABLE
POWERFUL FRONT PANEL SOFTWARE**



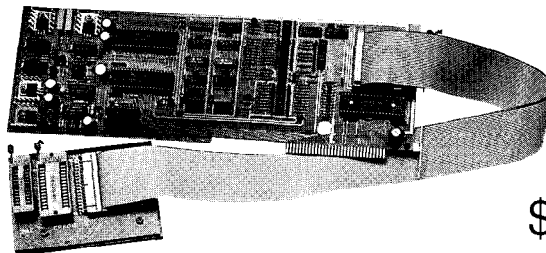
\$1799 - DSO-28204 (4K)
\$2285 - DSO-28264 (64K)

DSO Channels
2 Ch. up to 100 MSa/s
or
1 Ch. at 200 MSa/s
4K or 64K Samples/Ch
Cross Trigger with LA
125 MHz Bandwidth

Logic Analyzer Channels
8 Ch. up to 100 MHz
4K or 64K Samples/Ch
Cross Trigger with DSO

Universal Device Programmer

FAL
GAL
EPROM
EEPROM
FLASH
MICRO
PIC
etc..

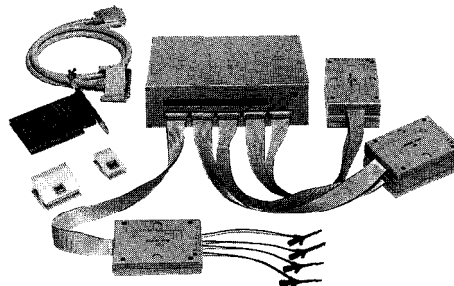


\$475

Free software updates on BBS
Powerful menu driven software

400 MHz Logic Analyzer

- up to 128 Channels
- up to 400 MHz
- up to 16K Samples/Channel
- Variable Threshold Levels
- 8 External Clocks
- 16 Level Triggering
- Pattern Generator Option



\$799 - LA12100 (100 MHz, 24 Ch)
\$1299 - LA32200 (200 MHz, 32 Ch)
\$1899 - LA32400 (400 MHz, 32 Ch)
\$2750 - LA64400 (400 MHz, 64 Ch)

Call (201) 808-8990

Link Instruments



369 Passaic Ave, Suite 100, Fairfield, NJ 07004 fax: 808-8786

FEATURE ARTICLE

Hank Wallace

No Emulator? Try a One-wire Debugger

A bug the night before a tradeshow drives Hank to invent his own debugger which scans for the smallest bit in a data burst and outputs the bitstream to the printer. He admits it's not an emulator, but in a pinch...



It was 10 P.M. and I was working on an 8051 product that I had designed and that a customer had requested some software changes for. Unfortunately, while making the changes, I introduced a bug. Being an economically paranoid designer, I had found a function for every pin on the micro, including the serial I/O. So, without an emulator, I was rather blind bug-wise.

In the past when I got into this situation, I have resorted to shifting the data out serially on an idle microprocessor I/O pin. On other occasions, I have even hung the program in a loop at a certain point and scoped the address lines to see if the micro hit the death point. None of these approaches is very programmer friendly or productive. They require a lot of squinting in dim light to visually capture a 1-MHz serial data burst on a nonstorage scope.

This frustration, however, produced an idea. I needed to expand my crude serial debugging method with some automation so it would be more useful in systems without explicit communication ability or an emulator. No doubt, many users of single-chip micros are in this situation.

I needed to send out some trace-point debug data, say, a few bytes, on one I/O line and capture it on a PC for display—this would enable me to view

critical system information. Of course, the serial-transmission routine had to be as small and unobtrusive as possible, transmitting data at whatever rate was convenient. Also, the polarity of the data had to be sensed by the PC and corrected accordingly so a developer could probe the datastream at any convenient point in the circuit.

SOLUTION OVERVIEW

After a lot of thrashing that night between 10 P.M. and 3 A.M., a final solution came out. The one-wire debugger has the following features and constraints:

- The target system shifts out its data prefaced by an 8-bit unique word. The value of this word is fixed at A5h and gives clock and polarity information to the PC program. (I did this so another I/O line would not be wasted for a clock signal.) An example of the 8051 routine is shown in Listing 1.
- The target can shift out a variable number of bytes and the PC figures out the rest. It displays data according to user specifications. In contrast to asynchronous serial data, there is no fixed-character formatting. The user tells the PC the length of the data burst.
- The data rate is not of much concern as long as it is between 150 Hz and 12 kHz (depending on the data length) and a 33-MHz 486 PC is being used as a baseline receiver. Although the PC adapts to the data rate, it is important that the data rate remain constant for the entire burst.
- The data sense can be inverted or true. Taking a cue from the polarity of the received unique word, the PC inverts the data if necessary.
- An output line that is typically static should be used because the decoding program is triggered by signal edges. The data represents only a quick disturbance to the output line, and there are typically some output lines in a system which would not be harmed by a fast data burst. For instance, the same product also has some output lines driving lamps which are typically in

one state. The lamps don't respond to the data.

- The PC uses one of its printer port's handshaking input lines to read the serial datastream at TTL levels.

HARDWARE CONFIGURATION

Figure 1 shows the debugger's hardware arrangement. I built a buffer out of a 4049 hex inverter to isolate my embedded system from the PC just in case of target meltdown, though this is not absolutely necessary if you can afford to toast your PC. But, I know, I know, it's 5 p.m., and the not-yet-working, trade-show demo system ships at 6 p.m., and the sales manager has been in your face for a week-use wire and a DB-25!

The input line at J2 is connected to your target system's temporary serial-data output and J1 is connected to the target's ground. The DB-25, P1, connects to your PC's parallel port. Power for the flea-power 4049 is derived from one of the PC's parallel-data output lines so the circuit can be switched off when not in use. This arrangement gives a high-impedance probe input and the ability to drive a few feet of cable without affecting the target system. The rest is software.

HOW IT WORKS

The PC program waits for a positive or negative edge in the datastream. It then samples data until the buffer is full or until no edges are seen for a while. The actual data samples are not stored, but only the duration of each high or low event. The program scans the data for 0-I-0 and 1-O-I noise glitches and deletes them.

The data is scanned again, looking for the smallest pulse width (assumed to be the width of one data bit). This smallest characteristic width always appears in the unique word. With this data bit width, the program averages

all other similar widths in the bitstream, arriving at an average bit duration. This duration is used to step through the bitstream and convert the samples to hard ones and zeros.

Once the data is converted, the unique word detector searches the data and, if the unique word is found, the remaining data is displayed. This whole process takes a fraction of a second, and after printing, the program recycles for another data burst.

Entering **RX ?** displays a list of command-line switches that the decoding program understands. These switches provide flexible formatting of data output including base conversions, byte, word, long integer, and ASCII modes, as well as time stamping, printer-port selection, and beep-on-decode alarm. If you need to run a test for an extended time to catch an infrequent bug, it will log data to a file. As well, the program has a framework for adding special formatting options as needed. Note that the L option for capturing data LSB first assumes that only the data is LSB first. The unique word must still be transmitted as A5h, MSB first, or 5Ah LSB first.

Another option, -Z, is included to allow testing of **RX**. C running on another PC. It causes **RX** to send repeatedly a fixed, four-byte datastream such as 1 1h, 22h, 33h, 44h. This datastream enables you to judge

the speed performance of **RX** on your PC without having to run any code in a target system. The data is emitted on pin 2 of the PC's DB-25 printer-port connector. That pin can be connected to the input of the hardware buffer for testing.

The 8051 routines in Listing 1 are used to dump data serially to the PC. Notice that interrupts are disabled during data output to ensure that the bit period is constant and not lengthened by interrupt activity. You

may leave interrupts enabled if the bit rate is low or the ISR execution time is small. The port line used here is P1.5. However, it should be changed to accommodate your target system.

The same is true for register usage. The bit-delay constants may also be adjusted for the master clock used in your target to get a usable bit rate. These routines are trivial to adapt to other micros.

This system does not decode data in real time, but rather buffers and analyzes a batch of data. Information learned about the latter part of a data burst is used to process the earlier part, something non-error-correcting, real-time decoders normally do not do.

Decoding data in real time is a more difficult problem. For bursty messages like this, though, decoding data offline provides simpler operation. This system is not optimized for use on noisy communications channels, like radio. So, beware!

The source code for this project is available on the Circuit Cellar BBS and can be modified for your needs. For instance, you may want to use other display formats or automated testing.

SYSTEM CONSTRAINTS

This debug method is meant for only short data bursts of just a few bytes, not for major core dumps. Use of longer bursts for light debugging of a

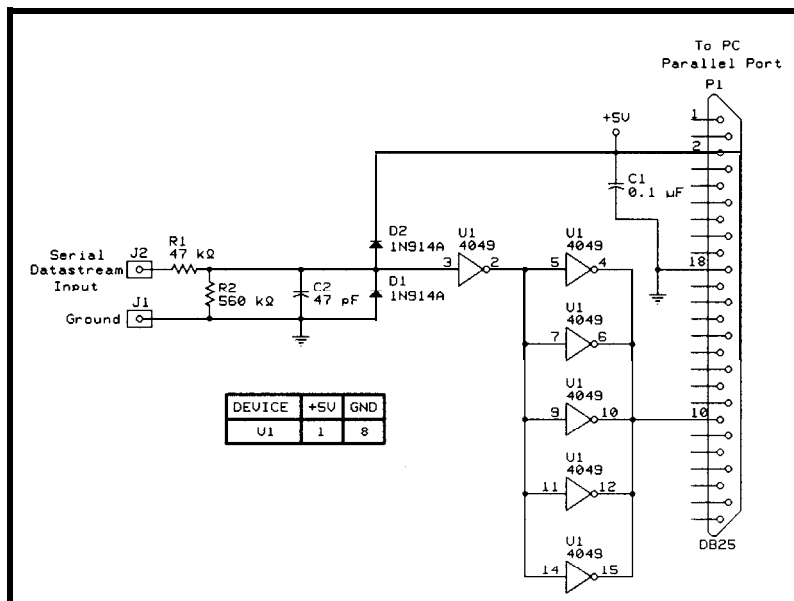


Figure 1—The one-wire debugger requires just a moment of time on an otherwise occupied I/O bit on the target system. The data is passed into a PC printer port, buffered by a 4049 (which is also powered by the printer port), and the rest is software.

target program seems unnecessary, but other applications may benefit.

One potential problem with the one-wire debugger is the timing drift which occurs as the data is decoded. It is caused by sampling granularity effects in the averaging phase; a decoder that adjusts its window during decoding would improve the situation. Another solution is to use a more complex encoding scheme which contains more clock information than the NRZ (nonreturn to zero) format used here. The FM format is such a scheme and is similar to that used in disk-drive data encoding. Encoding another format into the target system is, of course, more complicated. I ruled it out for this project around 11:15 P.M.

As a result of the problematic granularity, the decoder is overly sensitive to the data pattern at high data rates. For example, when decoding the unique word and four zero bytes, the only clock information available is contained in the unique word. By the time the decoder steps out into the fourth byte, some errors

Listing 1--The serial-debugging output routines for the 8051 can be easily adapted for other micros.

```

;void debug(int data) /* passed in r0,r1 */
; This function serially sends a unique word followed
; by the contents of r0 and r1.
;{
debug
mov     c,P1.5 ; tmp=P1.5;          /* save I/O bit state */
mov     psw.5,c
clr     IE.7 ; disable();          /* disable interrupts */
mov     a,#0a5h; shift_out(0xa5); /* send unique word */
acall  shift_out
mov     a,r0 ; shift_out(data & 0xff); /* send data */
acall  shift_out
mov     a,r1 ; shift_out(data >> 8); /* send data */
acall  shift_out
mov     c,psw.5; P1.5=tmp;        /* rest. I/O bit state */
mov     P1.5,c
setb   IE.7 ; enable();           /* enable interrupts */
acall  bitdly ; bitdly();          /* interburst delay */
ret

bitdly mov     r0,#120
L004   mov     r1,#255
L003   djnz   r1,L003
        djnz   r0,L004
        ret

;}
;void shift_out(char data) /* passed in accumulator */
; This function shifts out the byte passed, MSB first. A time
; delay is performed here to set the bit duration and should be
; adjusted for the clock rate used in the target system. This

```

(continued)

REMOTE POWER CARD!

3 VERSIONS:

RESET
WATCHDOG RESETS PC IF IT HANGS FOR HARDWARE OR SOFTWARE REASONS

PHONE
TURN ON PC WITH PHONE, SHARE VOICE / MODEM LINE, CONTROL AC APPLIANCES

TIMER
WAKEUP OR SHUTDOWN PC, LATE NITE BACKUP / MODEM, CONTROL AC APPLIANCES

95\$

27S OEM

ALL MVS CARDS INCLUDE ASM, BASIC, AND C SOURCE FOR PC OR SBC

8 CHAN ADC

DATA ACQUISITION, SERVO CTL, AUDIO
8-BIT RESOLUTION 22KHZ SAMPLE RATE
SHARP CUTOFF ANTI-ALIAS FILTER
CREATE STEREO BLASTER(.VOC) FILES

95\$

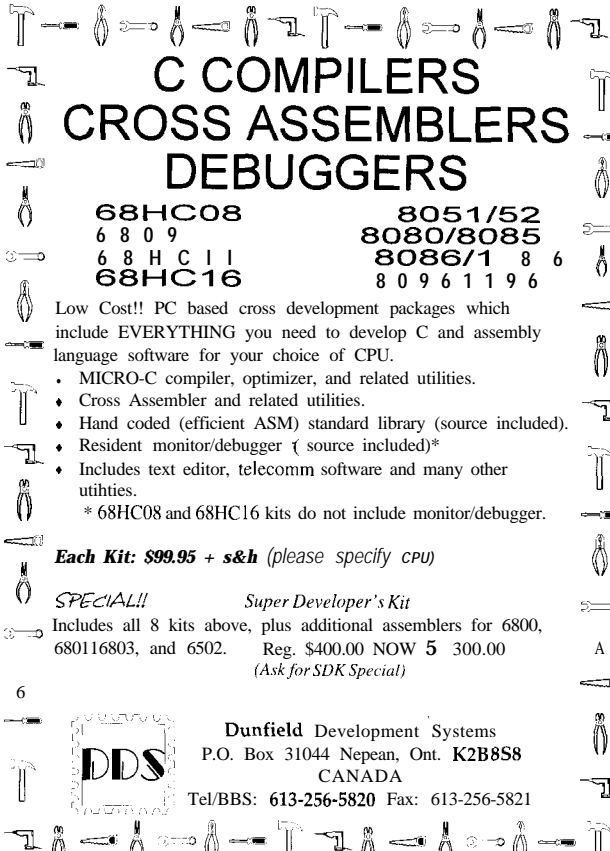
2 CHAN DAC

VOICE MAIL, ALARMS, CTL VOLT
8-BIT RESOLUTION 44KHZ SAMPLE RATE
PLAYS MONO / STEREO BLASTER FILES
FUNCTIONS AS DIGITAL ATTENUATOR TOO

75\$

5 YEAR LIMITED WARRANTY
FREE SHIPPING IN USA

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



C COMPILERS CROSS ASSEMBLERS DEBUGGERS

<p>68HC08 6809 68HC11 68HC16</p>	<p>8051/52 8080/8085 8086/186 80961196</p>
--	---


Low Cost!! PC based cross development packages which include EVERYTHING you need to develop C and assembly language software for your choice of CPU.

- MICRO-C compiler, optimizer, and related utilities.
- Cross Assembler and related utilities.
- Hand coded (efficient ASM) standard library (source included).
- Resident monitor/debugger (source included)*
- Includes text editor, telecomm software and many other utilities.

* 68HC08 and 68HC16 kits do not include monitor/debugger.

Each Kit: \$99.95 + s&h (please specify CPU)

SPECIAL!! Super Developer's Kit
Includes all 8 kits above, plus additional assemblers for 6800, 680116803, and 6502. Reg. \$400.00 NOW 5 300.00 (Ask for SDK Special)



Dunfield Development Systems
P.O. Box 31044 Nepean, Ont. K2B8S8
CANADA
Tel/BBS: 613-256-5820 Fax: 613-256-5821

Listing I-continued

```
; delay yields a bit rate of about 1800 bit/s with an 11.059.MHz
; crystal. Port P1.5 is used as the serial output line, and can
; be changed to suit your system.
;{
shift-out
  mov    r2,#8 ; i=8;
L001    ; do {
  mov    c,acc.7; P1.5=data & 0x80;
  mov    P1.5,c
  rl     a ; data<<=1;
  mov    r3,#255; for (x=255; x!=0; x--); /* bit delay */
L002    djnz  r3,L002; }
  djnz  r2,L001; while (--i != 0);
  ret
;}
```

will already have accumulated. Data may not be decoded properly if the rate is too high. On my '486/33 PC clone, this usability threshold is 1.5 kbps for 4-byte bursts, 3 kbps for 3-byte bursts, 6 kbps for 2-byte bursts, and 12 kbps for 1-byte bursts. Thus, if you need to blast out data quickly, keep it short.

Also, the mere printing of data takes time, and the receive software

does not scan for data while printing. There is a finite dead time while printing a data burst before the receiver is reinitialized. This means data bursts must be spaced apart in time somewhat (say, 100 ms), depending on the print options selected. Oh well, what did you expect for free?

NOT AN EMULATOR, BUT...

I hope the source code for this

project will make your 10 P.M. projects run a little smoother. Having used this technique, it still leaves me longing for a real emulator. But, for infrequent needs and a no-deep-pockets employer, this one-wire debugger works wonderfully! ☑

Hank Wallace is the owner of Atlantic Quality Design, an embedded systems hardware and software design firm located in Rural Hall, North Carolina. He can be contacted at (910) 377-2843 or hwallace@cybernetics.com.

SOFTWARE

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

I R S

404 Very Useful
405 Moderately Useful
406 Not Useful

ENGINEERING PROFESSORS

LISTEN UP!

The reason you read *Circuit Cellar INK* is obvious. Like thousands of others, you have come to realize that *Circuit Cellar INK* presents technically relevant computer-based applications in a comprehensive and instructive manner. In fact, our projects are presented just the way you'd do them if only you had the time.

In 1993, we provided thousands of your college students with industry quality development tools and information. We considered the program such a success that we'd like to extend our offer to the 1994-95 school year. But, spring semester is just around the corner ...

If you're an engineering professor or teach qualified technical students, we'd like to give you and all of your students free subscriptions to the *Circuit Cellar INK*. Please tell us how we can contact you directly so you don't miss a single issue.

WE WANT TO HELP AMERICA REGAIN THE COMPETITIVE EDGE. WE'RE HOPING YOU DO TOO!

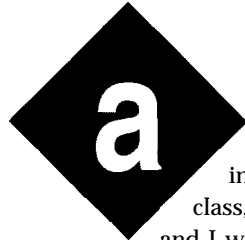
Rose Mansella, Circulation Coordinator
Circuit Cellar INK
4 Park St.
Vernon, CT 06066

Tel: (203) 875-2199
Fax: (203) 872-2204
Internet: rose.mansella@circellar.com

FEATURE ARTICLE

Steven Kubis

Using Spreadsheets to Simulate Digital Filters



as a project for an independent study class, another student and I were implementing an IIR digital filter using a 68HC11 microcontroller. We used MATLAB to design a second-order, low-pass IIR filter. MATLAB generated the transfer function that we implemented using the 68HC11. When we tested the filter, we found the output was sporadic.

Obviously, something was wrong with the filter, but we didn't know whether the problem was in the design or the implementation. To successfully troubleshoot the filter, we first had to verify that the design was correct. If we could do this, then we knew our problem was in the implementation, not the filter design.

Because it was a student project, our method had to be inexpensive. We decided to use a spreadsheet to simulate and verify the filter design.

IMPLEMENTING THE FILTER SIMULATION

We used Microsoft Excel for the Macintosh to implement the simulation. Any standard spreadsheet can be used. However, to be most useful, it's best if the spreadsheet can plot graphs directly on the worksheet (see Figure 1). To understand how the simulation works, let's first review digital filters.

Digital filters sample an input signal and calculate the output value

at specific, constant time intervals. The time between these intervals is the period. The sampling frequency of the filter is the reciprocal of the period. The characteristics of digital filters are based on the sampling frequency.

Spreadsheets work well for digital-filter simulation because instead of being periodic in time, they're periodic in position. Each row in the spreadsheet can represent one sample of the input signal and the resulting calculated output signal.

PARTS OF THE SPREADSHEET

The spreadsheet is composed of six parts:

- sample column—serves as the timebase for the simulated filter and is used as a basis for the input and output plot. The input signal for the simulated filter is derived using the sample column. For most simulations, 100 data points are adequate.
- input column—produces the simulated-input signal for the filter. The values in this column are computed based on the sample column and the values of the input signal frequency and sampling frequency. This is described in detail in the next section.
- filtered column—contains the output of the simulated filter based on the input signal and the filter coefficients.
- signal and filter characteristics—specify the input signal frequency and the sampling frequency of the filter. They also specify the input-signal magnitude and any offset.
- filter coefficients—come from the discrete-time transfer function used to describe the digital filter. These coefficients are used to calculate the values in the filtered column.
- input and output plot—enables the input signal and filtered output signal to be viewed. The plot is produced by plotting the values in the input and filtered columns versus the sample column.

PRODUCING THE INPUT SIGNAL

The input signal is the most difficult part of the spreadsheet to

Who says spreadsheets are just a financial tool? Steven proves otherwise. Cells easily accommodate input data and subsequent calculations of the output signal. Voilà, your spreadsheet is a digital filter!

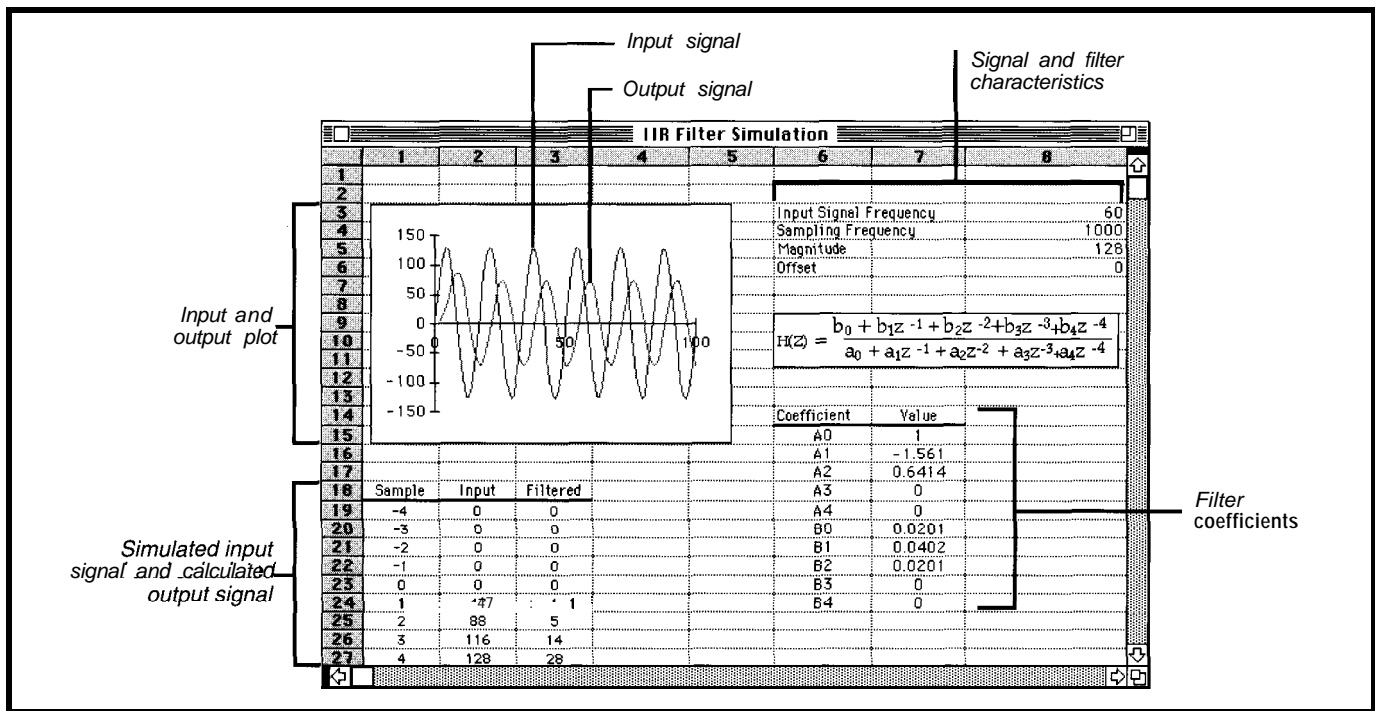


Figure 1—Microsoft Excel running on a Macintosh works well for filter simulation because it can display graphs of the data on the same screen as the spreadsheet itself.

implement. The input signal should be periodic and have characteristics of standard input signals (standard input signals include sine, square, and triangular waves). It also must demonstrate the correct relationship between the frequency of the input signal and the sampling frequency. Two basic calculations produce these results.

To be periodic, the number of samples in one period of the input signal must be calculated. This value is determined by the ratio of the sampling frequency and the input frequency.

$$\text{Samples Per Period} = \frac{\text{Sampling Frequency}}{\text{Input Frequency}}$$

One cycle of the input signal must be completed in this number of samples.

To correctly produce the input signal, the current position in the current period of the input signal must also be determined, as shown in Figure 2. The position in the period (PP) is calculated by:

$$PP = \text{Current Sample} \% \left(\frac{\text{Sampling Frequency}}{\text{Input Frequency}} \right)$$

where % is the modulus operator. Based on these calculations, the different input signals are produced (descriptions of some standard input

signals follow). Be sure to use absolute references for the input and sampling frequency, magnitude, and offset. Use a relative reference for the current sample.

The sine wave is simulated using the SIN () function found in standard spreadsheets. The signal is scaled based on the magnitude, rounded to the nearest integer, and then offset:

$$\text{Round} \left(\text{SIN} \left(\frac{PP \times 2\pi \times \text{Input Frequency}}{\text{Sampling Frequency}} \right) \times \text{Magnitude} \right) + \text{Offset}$$

The square wave is simulated using the IF () operation and COS () function found in standard spreadsheets. When the output of the COS () function is positive, the signal is the magnitude plus the offset. When the COS () function is negative, the signal is the offset less the magnitude:

$$\text{IF } \cos \left(\frac{PP \times 2\pi \times \text{Input Frequency}}{\text{Sampling Frequency}} \right) > 0 \text{ THEN} \\ \text{Magnitude} + \text{Offset} \\ \text{ELSE} \\ \text{Offset} - \text{Magnitude}$$

The triangle wave is simulated by two parametric equations. The signal starts at the positive magnitude and decreases to the negative magnitude by the middle of the period. In the second half of the period, the signal starts at the negative magnitude and increases

to the positive magnitude. The IF () operation determines which half of the period is currently being calculated (see Figure 3).

The output of the filter is calculated from the difference equation for the filter being simulated. The form of the difference equation varies depending on the type of filter. When entering the formula to compute the output, you should use absolute references to the filter coefficients. Using this method, you have to write only one formula, which can be copied to all rows in the output column.

USING THE SPREADSHEET

This spreadsheet can be used to simulate IIR filters, FIR filters, and

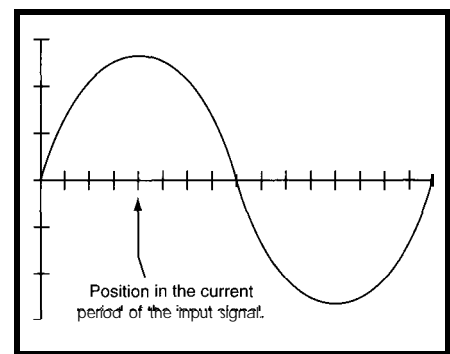
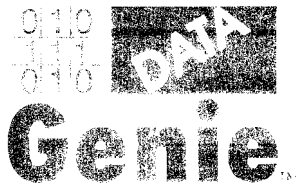
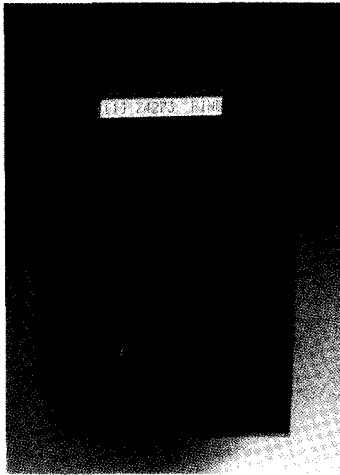


Figure 2—The current position in the current period of the input signal must be determined for the input signal to be periodic. In this illustration, the value of the input signal for the fourth of sixteen positions is calculated.



Data Genie offers a full line of test & measurement equipment that's innovative, reliable and very affordable. The "Express Series" of stand-alone, non-PC based testers are the ultimate in portability when running from either battery or AC power. Data Genie products will be setting the standards for quality on the bench or in the field for years to come.



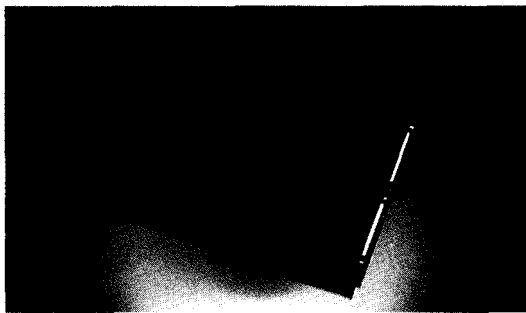
HT-28 Express
DIGITAL IC TESTER

The HT-28 is a very convenient way of testing Logic IC's and DRAM's. Tests most TTL 74, CMOS 40/45 and DRAM's 4164-414000, 44164-441000. It can also identify unknown IC numbers on TTL 74 and CMOS 40/45 series with the 'Auto-Search' feature.
\$189.95



HT-14 Express
EPROM PROGRAMMER

The HT-14 is one-to-one EPROM writer with a super fast programming speed that supports devices from 27328 to 27080. with eight selectable programming algorithms and six programming power (VPP) selections.
\$289.95



P-300
PC INTERFACE CARD PROTECTOR

The Data Genie P-300 is a useful device that allows you to quickly install add-on cards or to test prototype circuits for your PC externally. Without having to turn off your computer to install an add-on cards, the P-300 maintains complete protection for your motherboard via the built-in current limit fuses.
\$349.95

MING
Microsystems
Division of MING & P, INC.TM
17921 Rowland Street
City of Industry, CA 91748
TEL : (818) 912-7756
FAX : (818) 912-9598

Call for a dealer near you.
1-800-473-6606

Data Genie products are backed by a full 1 year limited factory warranty.

analog filters that have been converted to discrete-time form. I suggest you make a template for each type of filter and input, then begin experimenting. The description of three filters follow. Use these filter designs to test your templates.

The following equation is the generalized transfer function for an IIR filter of order q .

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{b_0 + b_1z^{-1} + \dots + b_qz^{-q}}{a_0 + a_1z^{-1} + \dots + a_qz^{-q}}$$

In Figure 4, we see what converting this to a difference equation yields. The value of the difference equation is computed to produce the filter output. When calculating the difference equation, it's a good idea to use as many coefficients as the highest-order filter needs. When simulating lower-order filters, enter zeros for the higher-order coefficients.

Note that the output is calculated from the current input and previous input and output values. For the filter to be causal, you need to use zero as the input for the samples prior to sample 1. The number of these zero samples depends on the order of the filter you're simulating. Also, when you write the output formula, be sure to use relative references to the previous input and output values.

After you've created the template for an IIR filter, try simulating the filter shown in Figure 5a. This filter is a second-order, low-pass, Butterworth filter, designed using MATLAB for a sampling frequency of 1000 Hz and a cutoff frequency of 50 Hz.

The following is the generalized transfer function for an FIR filter of order q .

$$H(Z) = \frac{Y(Z)}{X(Z)} = a_0 + a_1z^{-1} + a_2z^{-2} + \dots + a_qz^{-q}$$

Converting this to a difference equation yields:

$$y(n) = a_0x(n) + a_1x(n-1) + \dots + a_qx(n-q)$$

Again, it's the difference equation that's calculated to produce the output. The difference equation is much simpler for a FIR filter, but FIR filters must be of a much higher order to have the desired characteristics. This makes FIR filters less practical to

```

IF PP ≥ ( Sampling Frequency / ( 2 × Input Frequency ) ) THEN
  Magnitude = [ ( ( 4 × Input Frequency × Magnitude ) / Sampling Frequency ) × ( Current sample % ( Sampling Frequency / ( 2 × Input Frequency ) ) ) ] + Offset
ELSE
  - Magnitude + [ ( ( 4 × Input Frequency × Magnitude ) / Sampling Frequency ) × ( Current sample % ( Sampling Frequency / ( 2 × Input Frequency ) ) ) ] + Offset

```

Figure 3—The formula to simulate a triangle-wave input signal uses an IF () operation to determine which half of the period is currently being calculated.

simulate using a spreadsheet, though it can be done.

After you've created the template for the FIR filter, try simulating the filter given in Figure 5b. This filter is a tenth-order notch filter designed with

$$y(n) = \frac{b_0x(n) + b_1x(n-1) + \dots + b_qx(n-q) - [a_1y(n-1) + \dots + a_qy(n-q)]}{a_0}$$

Figure 4—A difference equation is used to calculate the output of an IIR filter of order q.

MATLAB for a sampling frequency of 1000 Hz. It has a center frequency of 125 Hz and a bandwidth of approximately 100 Hz.

Analog filters can be simulated if they have been converted to a discrete-time form. The trapezoid rule is a good conversion to use to transform an analog filter from continuous-time

```

a)
a0 = 1.0000 a1 = -1.5610 a2 = 0.6414
b0 = 0.0201 b1 = 0.0402 b2 = 0.0201

b)
a0 = -0.0416 a1 = -0.1886 a2 = -0.0852
a3 = -0.0024 a4 = 0.1268 a5 = 0.1809
a6 = 0.1268 a7 = -0.0024 a8 = -0.0852
a9 = -0.1886 a10 = -0.0416

```

Figure 5—Different kinds of filters—such as a second-order, low-pass IIR filter (a) and a tenth-order, notch FIR filter (b)—can be simulated by inserting different coefficients.

form to discrete-time form. The conversion is shown as:

$$\frac{2(z-1)}{T(z+1)}$$

where T is the sampling period. After substituting for s in the continuous-time form and selecting a sampling period, simplify the expression and find the difference equation. Typically, the discretized transfer function has a difference equation like an IIR filter.

As an example, try simulating the filter shown in Figure 6. It has a reso-

nant frequency of 100 Hz, and the output is the voltage across the resistor. The filter's transfer function is:

$$G(s) = \frac{10}{10 + 0.2533s + \frac{100000}{s}}$$

Discretized using the trapezoid rule and a 1 000-Hz sampling frequency, the discrete-time form is:

$$H(Z) = \frac{1-z^2}{56.66 - 91.322z^{-1} + 54.622z^{-2}}$$

You can use the template you created for the IIR filter to verify the characteristics of this analog filter.

EXPERIMENTATION

After your templates work correctly, you can begin to experiment with the items listed below:

- step response and settling time—make a template with a unit step input to view the step response and estimated settling time.
- aliasing—watch for aliasing to occur when the sampling frequency is too low for the input signal.
- output signal quality—note that the quality of the output signal degrades as the input signal approaches the sampling frequency.

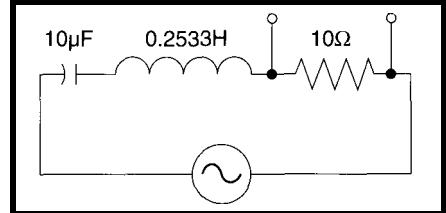


Figure 6—One example of an analog filter that can be simulated is a band-pass filter with a resonant center frequency of 100 Hz. The output is across the resistor.

• unstable filters—add an extra term to the denominator of the example IIR filter. This will add an unstable pole to the filter, causing the output to “explode.” (Fortunately, in the simulation, this just means extremely large output values, not

- noise—use the RAN D () function in your spreadsheet to add simulated noise to the input signal.

The only limits are the spreadsheet's capabilities and your own creativity.

CONCLUSION

Basic digital filters can be simulated using a spreadsheet, a tool most people already have. Once you've created a set of templates, the simulations are easy to use, flexible, accurate, and best of all inexpensive.

Getting back to my original problem, the spreadsheet simulation showed that the filter design was correct. The problem was in the hardware. After some debugging, the filter worked like the design. It was a beneficial problem, though. Now my colleague and I know that spreadsheets aren't just a financial tool. 📌

Steven Kubis is currently a senior technical writer with Great Plains Software in Fargo, North Dakota. He graduated in 1993 with a BSEE degree from North Dakota State University. He may be reached at skubis@cogs.gps.com.

REFERENCES

R. E. Ziemer, W. H. Tranter, and D. R. Fannin, **Signals and Systems: Continuous and Discrete**, New York: Macmillan Publishing Company, 1989.
The Student Edition of MATLAB Reference Manual, Englewood Cliffs: Prentice Hall, 1992.

I R S

- 407 Very Useful
- 408 Moderately Useful
- 409 Not Useful

FEATURE ARTICLE

Art Sobel

A RISC Designer's New Right ARM Writing Code for the ARM Processor

After an introduction to the ARM and its hardware in the October and December issues, Art wraps up his three-part series with the fundamentals of ARM software. Listen in for some good "how tos."



As you may recall, the ARM processor first resided on a plug-in, second-processor board in the 6502-powered BBC computer and relied on the host computer to run the file system and user interface.

The satellite board had a small supervisor program called the *Brazil Monitor*, which mediated communication with the host and enabled ARM programs to pretend that they were on the main computer. Operating system calls were made with the software interrupt (SW I) instruction, similar to the way that the PC uses the INT instruction for DOS and BIOS calls. The assembler and compilers for the ARM form the basis of the current ARM toolkit.

In 1987, Acorn made its first ARM-based computers. It extended the Brazil Monitor and added all the functionality of the BBC host machine including a BASIC interpreter, equivalent BBC file system, and a desktop user interface.

This new operating system was called *Arthur*.

The newer versions of the ARM operating system are called by the more ordinary name of RISC-OS. In typical, conservative software fashion, RISC-OS retains all of the operating system calls of the preceding programming environments—Brazil, BBC, and Arthur.

Unlike a DOS machine, Acorn computers have both the BIOS functions and the operating system in ROM. The ROM also contains file system support for floppies in Acorn and DOS formats; IDE disk drivers; drawing routines (similar to QuickDraw and Display PostScript); window, font, memory and task managers; and BASIC interpreter and editor.

Greater functionality can be added through the use of modules which are loaded from disk and reside in RAM. Modules provide new SW I calls that extend the operating system. This method is similar to a TSR (terminate and stay resident) program on a PC. Plug-in boards also add their own drivers and operating-system extensions.

When the ARM600 was built in 1991, the first version of the cross-development software was also released. Currently, the ARM Cross-Development Toolkit includes the assembler, C compiler, linker, libraries, and support utilities. The tools let you develop, test, and refine embedded ARM applications using a PC-compatible computer or UNIX workstation. The toolkit is distributed to developers by most of the ARM licensees.

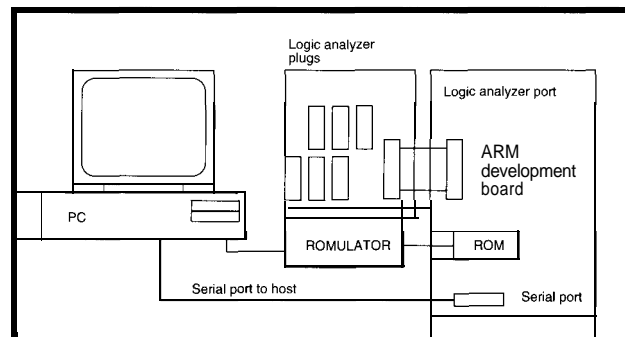


Figure 1—The basic setup for developing ARM software uses a host computer (PC or UNIX) to develop code and to operate the debugger. Optionally a logic analyzer is useful to monitor board operation. A ROM emulator downloads code to ROM sockets when building ROM images

Table 1—This is the register assignment that is standard for C using the armcc compiler. When a project uses mixed C and assembler, adhering to this standard ensures that code segments work together. In general, assembler programs must preserve all registers above R3, retrieve arguments, and return values in the first four registers.

DEMON-ROM MONITOR

ARM also developed the DEMON monitor and debugger program to operate with their cross-development toolkit. It was a direct descendant of the Brazil Monitor, although it was extensively rewritten and modularized for easy porting. As I mentioned in the last article, great care was taken to ensure that DEMON worked the same on the PIE ARM60 demo board and the PID ARM600 development board.

Of course, DEMON has already started to undergo a new round of modifications. The trouble started when the μ C/OS real-time kernel was ported to the ARM. In its original form, DEMON was useful in loading and debugging single-threaded demo or user programs. With a real-time tasking kernel, it interfered with the RTOS operation so that the two could not function at the same time. Debugging was very laborious. Geary Chopoff rewrote the DEMON to be compatible with the μ C/OS kernel, and converted much of it to C.

GNU TOOLS-NOT UNIX OR ANYTHING ELSE

GNU software tools from the loosely organized and named free-software foundation have been used by many processor manufacturers as the basis for their software development. GNU has a C compiler, assembler, linker, and debugger available from many sources in both binary and source code.

This past summer, work was done to add to GNU the capability of generating ARM code from C. GAS (GNU Assembler) was also modified to support ARM assembler text input and standard object-format output.

APPLE NEWTON

Apple also uses the ARM cross-development toolkit for Newton

Reg.	Assign.	Use
R0	a1	arg. 1/int. result/scr. reg.
R1	a2	arg. 2/scratch register
R2	a3	arg. 3/scratch register
R3	a4	arg. 4/scratch register
R4	v1	register variable
R5	v2	register variable
R6	v3	register variable
R7	v4	register variable
R8	v5	register variable
R9	sb/v6	static base/reg. variable
R10	sl/v7	stack limit/stack chunk handle/reg. var.
R11	fp	frame pointer
R12	ip	low end of cur. stk frame
R13	sp	scratch reg./new-sbininter-link-uit calls
R14	lr	link address/scratch reg.
R15	pc	programcounter

development. The toolkit runs in the MPW (Mac Programmers Workbench) environment and couples to the Newton through the AppleTalk serial port. The Newton OS is a unique operating system with no relationship to the Acorn OS.

For instance, data storage in the Newton is not based on files, but on a unified object structure which allows any data to be accessed by any application. Newtonians have gotten around this by renaming files as "soups of frames of objects." Media-like flash cards have become "collections of soups." Applications are organized into hierarchies of "templates" that contain descriptions of fields; graphic objects, buttons, and attached scripts (methods); and finally other templates.

The preferred Newton programming language, NewtonScript, is an object-oriented dynamic language in which object binding is done on the fly like SmallTalk and not statically like C++. NewtonScript defines templates and other kinds of data, and is used to retrieve and store data, query the I/O and touch screen, and call C and assembler routines for special or accelerated functions.

The Newton operating system also supports preemptive multitasking and

Listing 1--The C and corresponding assembler code for a simple "Hello World" program illustrate a bit of the ARM's instruction set.

```
#include <stdio.h>
int main(int argc, char ** argv)
{
    printf("Hello World \n");
    return 0;
}

;hellow.lst generated by Norcroft ARM C vsn 4.50
1 00000000
2 00000000 AREA |C$$code|, CODE, READONLY
3 00000000 |x$codeseg|
4 00000000
5 00000000 6D 61 69 6E DCB &6d,&61,&69,&6e
6 00000004 00 00 00 00 DCB &00,&00,&00,&00
7 00000008 FF000008 DCD &ff000008
8 0000000c
9 0000000c IMPORT printf
10 0000000c EXPORT main
11 0000000c min
12 0000000c E1A0C00D MDV ip,sp
13 00000010 E92DD803 STMDB sp!,{a1,a2,fp,ip,lr,pc}
14 00000014 E24CB004 SUB fp,ip,#4
15 00000018 E28F0008 ADD a1,pc,#L000028-.-8
16 0000001c EBF00007 BL printf
17 00000020 E3A00000 MDV al,#0
18 00000024 E91BA800 LDMDB fp,{fp,sp,pc}
19 00000028 1000028
20 00000028 48 65 6C 6C DCB "Hello World \n"
21 0000002c 6F 20 57 6F
22 00000030 72 6C 64 20
23 00000034 0A 00 00 00
24 00000038 AREA |C$$data|,DATA
25 00000000 |x$dataseg|
26 00000000 END
```

ARM DEVELOPMENT

Let's take the standard program "Hello World" as a short example (compiled with `armcc`):

```
C:>armcc -li-apcs 3/32bit -S hellow.s -o hello.o hellow.c
```

The command generates an assembly and object file for us to look at. To get the assembly listing we assemble the `hellow.s` file with:

```
C:>armasm -li hellow.s -list hellow.lst
```

From the `hellow.o` file, we can get the executable file (`hellow`) by using the `arm1ink` program and the appropriate library file. As with most C compilers, the `hellow` file is quite large because of the inclusion of `printf` from the library.

```
C:>arm1ink -o hellow hellow.o Software/lib/arm1ib.321
```

Now we start the debugger.

```
C:>armsd hellow
```

The debugger displays a logon banner identifying critical information. The list command can be used to display the program. When you execute the program, the program lists:

```
armsd: go
Hello World
Program terminated normally at PC = 0x00009f64
+0024 0x00009f64: 0xef000011 : swi 0x11
armsd:
```

The most useful functions of the debugger are:

```
Load <imagefile> [<arguments>]
List [<expr1>[, [+>expr2>]]
Break [<context>[<count>][D0'!'<command>;]'] [IF<expr>]]
```

```
Examine [<expr1>[, [+>expr2>]]
Registers [mode]
```

`Load` loads an image for debugging. `<imagefile>` is the filename of the image and `<arguments>` are any command line arguments expected by `<imagefile>`.

`List` examines memory contents in instruction, hex, and character format. If "+" is specified, `<expr2>` is a byte count.

`Break` sets a breakpoint or, with no arguments displayed, gives the breakpoint list. `<count>` specifies the number of times the breakpoint must occur before execution is halted or `<expr>` is tested. When the optional "IF" clause is specified, execution is only halted when `<expr>` evaluates to nonzero. If the optional "D0" clause is specified, then the commands enclosed in the braces are executed when program execution is stopped because of the breakpoint. Code can step by source program statements as directed. For instance, `in` directs it to step into calls, `<count>` specifies the number of statements or instructions to be stepped, and `<expr>` specifies a condition which must evaluate to 0 before stepping stops.

`Examine` checks memory contents in hex and character format. If the "+" is specified, `<expr2>` gives a byte count.

`Registers` displays the contents of ARM registers R0-R15 of the current mode and decodes the PSR. If a mode is given, display the contents of those registers which differ between the named and the current mode.

Although the debug support software is currently command-line driven, it is being expanded into a full GUI debugger called *JumpStart*, which should be available for the PC in early 1995.

memory protection using the MMU of the ARM610. Apple also uses the ARM in big-endian mode (byte 0 corresponds to D[31:24]) instead of the Acorn-preferred little-endian mode. It is not likely that this software will be used by independent programmers for their embedded applications since it is entirely in ROM and not divided into convenient OS and BIOS partitions. As a final factor, Apple also strictly controls who gets licenses for its software.

HELIOS REAL-TIME OS

Real-time operating systems based on Micro-kernel architecture are the current rage. Fortunately, this type of operating system has just been announced by Perihellion Distributed Software and is called *Helios/ARM*.

Helios operates like UNIX with real-time extensions and has a POSIX interface so that many programs written for UNIX workstations work

on ARM development boards after recompiling. A version of Helios has already been ported to the PID. When an SMC Ethernet board is added, the

PID appears as a UNIX node on a TCP/

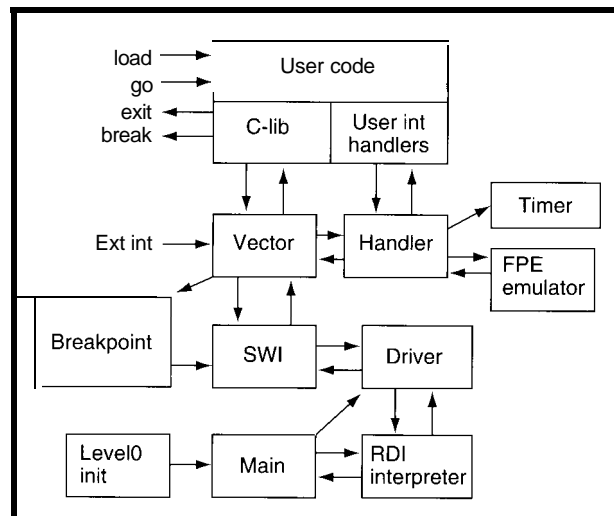


Figure 2—Within the main module of C-DEMON, Level0 starts when the board comes out of reset and initializes the RAM and DEMON data structures. The main program operates as a loop, waiting for communication from the host computer through the driver routine. This communication is interrupted and acted on by the RDI interpreter. Interrupts are directed through the vector module and then the handler, which then redirects the interrupt routine to user code.

Top of RAM	
8000	User software
1000	Floating-point emulator
0A00	Command line
0B00	Other DEMON variables
0A00	Driver variables
0284	Processor stacks
00D4	Breakpoint support
00B4	MPU vector area
0094	FIQ vectors
0074	IRQ vectors
0064	Misc. support functions
001C	Breakpoint vector area
0000, 0000	MPU vector area

Figure Z-C-DEMON expects a specific *memory organization*. The MPU exception vectors are preassigned by hardware. Breakpoint vectors are in the lowest page because of the use of direct PC loading. Soft vector facilities, which can be reassigned by the user, include processor stacks (one for each mode), global variables, and the floating-point emulator.

IP network! This is sophisticated stuff with a lot of bytes.

Although the first versions of Helios for ARM have begun to ship, work remains on the multithreaded debugging environment. We expect that Helios with the multithreaded debugger will appear by June 1995 and X Windows by Christmas.

At this time, Helios is too complicated to be adequately covered in a short article. I will cover it when it has a specific implementation such as in an ARM7500 embedded computer board.

ARM SOFTWARE DEVELOPMENT

Figure 1 shows the typical development setup used with a NP1E [VLSI's version of PIE) or PID board. As with many embedded CPUs covered in **Circuit Cellar INK**, there is no native development environment available for the ARM (at least in the U.S.). Program writing, compiling, and assembling must be done on a separate workstation or PC. The PC connects to the development board through a serial port and a null-modem cable.

TEAM PARADIGM

We've beefed up our well-respected arsenal with new releases featuring more capacity and powerful new capabilities. Experience the raw power and searing speed of these field-tested development tools **vpurself**. Be ready to kick a little butt and rest assured that **Team Paradigm** is here to back you up.



NEW!
DEBUG 4.0

HEY DUDES! CHECK OUT THE NEW PARADIGM DEBUG 4.0. IT'S A KILLER APP WITH THE RIGHT BALANCE FOR FINDING AND EXTERMINATING PESKY BUGS. NOTHIN' LIKE RAW FIREPOWER FOR GETTIN' THE JOB DONE -- THE FIRST TIME.

I'VE GOT THE PERFECT DEBUG COMPLEMENT. YEAH, MR. 'SEARCH & DESTROY' IS A REAL BUG KILLER, BUT I NEED MAXIMUM AGILITY AND FINESSE. NOTHING MEASURES UP IN HANDLING THE MOST DIFFICULT C/C++ EMBEDDED APPLICATIONS.

NEW!
LOCATE 5.0

'Nuff said.

PARADIGM

Proven Solutions for Embedded C/C++ Developers
1-800-537-5043

Paradigm Systems
3301 Country Club Road, Suite 2214
Endwell, NY 13760
(607) 748-5966
FAX: (607) 748-5968
Internet: 73047.3031@compuserve.com

TO BE CONTINUED...

The serial port can speed along at 38.4 kbps, but large programs still take a while to download.

If the ROM is being debugged (as when developing a new version of the DEMON), the use of a ROM emulator is highly recommended. The NP1E operates out of a single 1-Mb ROM, while the PID requires four 256-Kb ROMs to include the whole DEMON. We have included a 32-bit logic-analyzer port on the development cards. When tracing the operation of a new board or ROM, this can be a great time saver. Both HP and Fluke logic analyzers have been used.

After writing code in assembler or C, the source is converted to ARM-object format (AOF) using the ARM assembler or C compiler. At this time, syntax or typing errors such as dangling labels are fixed. When mixing assembler and C modules, the programmer is encouraged to use a common format for software called APCS (ARM Procedure Call Standard). Table 1 outlines some of the APCS standards.

Listing 2—Interrupt handlers *must* be installed before they can be used by user code. The assembler version (a) explicitly uses SWI 0x70 (InstallHandler software interrupt call) while the C version (b) uses a call to a preassembled handler that then uses the same SWI call.

```

a)
InstallHandler EQU 0x70
      MOV a1,#vecnum      ;place vector num in r0
      LDR a2,#0           ;place vector value in r1
      LDR a3,#vecloc      ;addr of int routine in r2
      SWI InstallHandler ;install the new vector
      CMP a2,#NULL       ;installed?
      BEQ NotInstalled   ;if NULL, error
      LDR a1,#NewLoc     ;save new vector loc
      STR a2,[a1]        ;then the previous
      LDR a1,prevvec     ;vector for restoration
      STR a3,[a1]        ;when done

```

```

b)
retval = SWI_InstallHandler(0x21, 0, vecloc);
if (retval.p2==NULL)
    printf("Error: Unable to install");
NewLoc = retval.p2; /* save new vector loc */
prevvec = retval.p3; /* remember for restore */

```

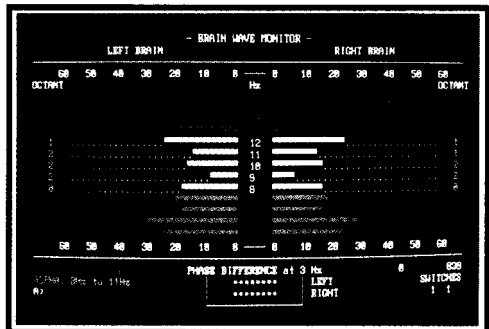
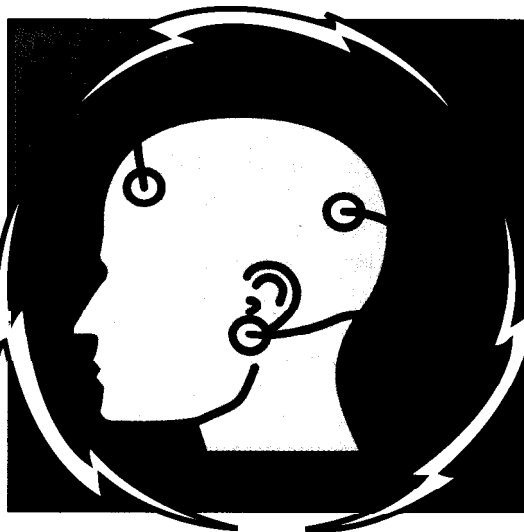
ARM objects are linked together or to any standard C library using a r m i n k. Linking resolves external references and outputs a variety of formats. AIF (ARM interchange format) is used for loading through the

debugger while binary format is used with the EPROM programmer or ROM emulator. The resultant executable code can be tested with the a r m s d debugger either through software emulation or testing on the target

HAL - 4

EEG Biofeedback Brainwave Analyzer

The HAL-4 kit is a complete battery-operated 4-channel electroencephalograph (EEG) which measures a mere 6" x 7". HAL is sensitive enough to even distinguish different conscious states-between concentrated mental activity and pleasant daydreaming. HAL gathers all relevant alpha, beta, and theta brainwave signals within the range of 4-20 Hz and presents it in a serial digitized format that can be easily recorded or analyzed. HAL's operation is straightforward. It samples four channels of analog brainwave data 64 times per second and transmits this digitized data serially to a PC at 4800 bps. There, using a Fast Fourier Transform to determine frequency, amplitude, and phase components, the results are graphically displayed in real time for each side of the brain.



HAL-4 KIT.....NEW PACKAGE PRICE - \$279 +SHIPPING
Contains HAL-4 PCB and all circuit components, source code on PC diskette, serial connection cable, and four extra sets of disposable electrodes.

to order the HAL-4 Kit or to receive a catalog,
CALL: (203) 8752751 OR FAX: (203) 875-2204

**CIRCUIT CELLAR KITS • 4 PARK STREET
SUITE 12 • VERNON • CT 06066**

• The Circuit Cellar Hemispheric Activation Level detector is presented as an engineering example of the design techniques used in acquiring brainwave signals. This Hemispheric Activation Level detector is not a medically approved device, no medical claims are made for this device, and it should not be used for medical diagnostic purposes. Furthermore, safe use requires HAL be batten, operated only!

board. This process uncovers additional errors in design and coding. After several such cycles, the code is deemed adequate and shipped.

The ARM development sidebar contains an overview of program development using armcc, armasm, and armlink on the familiar helloworld program (shown in Listing 1).

THE C-DEMON

The C-DEMON is VLSI's version of ARM's DEMON (Debug Monitor). Although DEMON is written in

assembly, C-DEMON is mostly written in C. Both interface to the ARM debugger using RDP/RDI (Remote Debug Protocol/Remote Debug Interface) over a serial communications line using RS-232 at a default speed of 9600 bps. This protocol is buried inside the host-based armsd program so that the user only sees intelligible commands. C-DEMON offers either a command-line or graphical debugger.

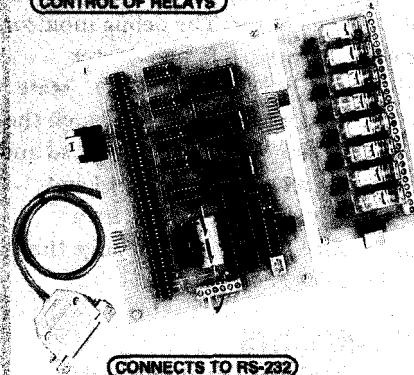
Figure 2 shows the relationship between major functional blocks of the

SWI	Num	C Prototype and Description
WriteC	0x00	void SWI_WriteC(int ch) Write char ch to console DOS call: Display Character INT21 h ah=2h DL=character
Write0	0x02	void SWI_Write0(char *cp) Write null-terminated string to console DOS call: PrintString INT21 h AH=9 DS:DX=character pointer
ReadC	0x04	unsigned int SWI_ReadC(void) Read char from console DOS call: INT21 h AH=1AL=returned character
CLI	0x05	void SWI_CLI(char *cp) Pass string pointed at host's CLI. No DOS equivalent
Exit	0x11	void SWI_Exit(void) Done with program (or process) Like DOS call: INT21, ah=31 h, terminate and stay resident Monitor erases program when new one is loaded over it
EnableINT	0x13	User program enables IRQ. DOS uses STI instruction
DisableINT	0x14	User program disables IRQ. DOS uses CLI instruction
EnterOS	0x16	void SWI_EnterOS(void) Enter SVC (supervisor) mode
GetErrno	0x60	unsigned int SWI_GetErrno(void) Get value of <errno> in r0 DOS Extended Error Info: INT 21 h AH=59h
Clock	0x61	unsigned int SWI_Clock(void) Read the system clock DOS call: INT 21 h AH=2Ch nearest equivalent
Time	0x63	unsigned int SWI_Time(void) UNIX number of seconds since 1/1/70
Remove	0x64	unsigned int SWI_Remove(char *cp) Remove filename in ASCII format
Rename	0x65	unsigned int SWI_Rename(char *old, char *new) Rename old to new (both ASCII) r0 and r1 pointers
Open	0x66	unsigned int SWI_Open(char *fn, int mode) Open filename in ASCII to mode -r0 pointer and r1 mode
GetVector	0x67	void *SWI_GetVector(int vecnum) Get the addr location of vecnum vector
Close	0x68	unsigned int SWI_Close(unsigned int handle) Close file by handle -r0 has handle number
Write	0x69	unsigned int SWI_Write(unsigned int handle, char *buf, int num_bytes) Write num_bytes from buf to file by handle
Read	0x6A	unsigned int SWI_Read(unsigned int handle, char *buf, int num_bytes) Read num_bytes from file by handle to buf
Seek	0x66	unsigned int SWI_Seek(unsigned int handle, unsigned int pos) Move pointer into file by handle to location at pos
Flen	0x6C	long int SWI_Flen(unsigned int handle) Get file length of file by handle (or -1 if unable to)
IsTTY	0x6E	int SWI_IsTTY(unsigned int handle) Returns 1 if file is TTY, else return 0
TmpNam	0x6F	unsigned int SWI_TmpNam(char *buf, int buflen) Get temporary filename from OS

Figure 4—The user software communicates with the host through software interrupts as in the PC. Just a sample of the available BIOS and OS functions are included in the SWI list.

RELAY INTERFACE

PROVIDES SOFTWARE CONTROL OF RELAYS

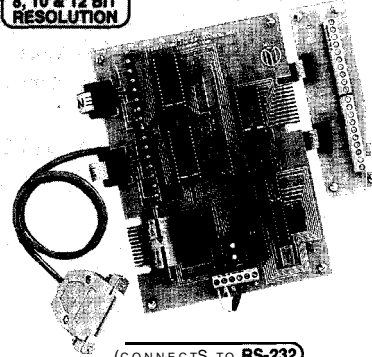


CONNECTS TO RS-232

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

ANALOG TO DIGITAL

8, 10 & 12 BIT RESOLUTION



CONNECTS TO RS-232

ADC-16 A/D CONVERTER* (16 channel/8 bit).....\$ 99.95
ADC-8G A/O CONVERTER* (8 channel/IO bit).....\$124.90
Input voltage, amperage, pressure, energy usage, joysticks and a wide variety of other types of analog signals. RS-422/RS-485 available (lengths to 4,000'). Call for info on other A/D configurations and 12 bit converters (terminal block and cable sold separately).
ADC-8E TEMPERATURE INTERFACE* (8 ch).....\$ 139.95
Includes term. block & 8 temp. sensors (-40° to 146° F).
STA-9 DIGITAL INTERFACE* (9 channel).....\$ 99.95
Input on/off status of relays, switches, HVAC equipment, security devices, smoke detectors, and other devices.
STA-8D TOUCH TONE INTERFACE*\$ 134.90
Allows callers to select control functions from any phone.
PS-4 PORT SELECTOR (4 channels RS-422)....\$ 79.95
Converts an W-232 port into 4 selectable RS-422 ports.
CO-485 (RS-232 to RS-422/RS-485 converter).....\$ 44.95

*EXPANDABLE...expand your interface to control and monitor up to 512 relays, up to 576 digital inputs, up to 128 analog inputs or up to 128 temperature inputs using the PS-4, EX-16, ST.32 & AD-16 expansion cards

* FULL TECHNICAL SUPPORT...provided over the telephone by our staff. Technical reference & disk including test software & programming examples in Basic, C and assembly are provided with each order.

* HIGH RELIABILITY...engineered for continuous 24 hour industrial applications with 10 years of proven performance in the energy management field.

* CONNECTS TO RS-232, RS-422 or RS-485...use with IBM and compatibles, Mac and most computers. All standard baud rates and protocols (50 to 19,200 baud).

Use our 800 number to order FREE INFORMATION PACKET. Technical Information (614) 464.4470.

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

International Ei Domestic FAX (614) 454-9656
Use for information, technical support & orders

ELECTRONIC ENERGY CONTROL, INC.
380 South Fifth Street, Suite 604
Columbus, Ohio 43215-5438

C-DEMON. Figure 3 shows the low-memory map of the PID with some detail on the areas used for the DEMON monitor. The debug monitor provides information on register values, processor mode, and the state of the memory locations. Through the RDI-byte commands, you can read and write a memory location, read and write a register, read and write a coprocessor's register, or change the mode or flags.

BREAKPOINTS

The DEMON would be useless without a breakpoint. To set a breakpoint, you must be able to change the code at the location of the breakpoint. Any location in RAM used for code can have a breakpoint, but you should not try to set one in a field of data since only opcodes can be executed.

To set breakpoints, C-DEMON takes advantage of the ARM instruction set, which allows the program counter or R15 to be set to an immediate value. Developers of ARM were familiar with the 6502 from MOSTEK, which had a range of instructions that reference a zero page. With the `MOV pc, #immediate`, we have a similar structure which can be used for operating system calls. The immediate value becomes a pointer to the range of 0x000-0x0FF or 0x000-0x3FC when the immediate is shifted left by four. By judicious partitioning, you can assign pointers to vectors that accomplish a breakpoint with a single instruction.

With PID interrupt handlers, both the basic FIQ and IRQ exception vectors can be replaced. But most times, it is the particular subevent which generated the FIQ or IRQ that we wish to observe. The PID has eight events associated with the FIQ and another eight with the IRQ. These events correspond to the 8-bit FIQ and IRQ status registers in the INTWT PGA. There are two tables set up in

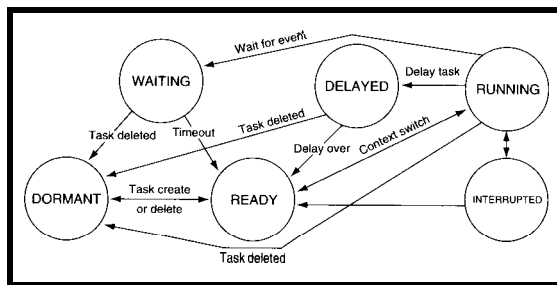
IRQ vectors and **FIQ vectors**. When initialized, they point to a do-nothing-return location.

For the interrupt to do work, it must be hooked to the start location of your program's interrupt handler. You can do this by:

μC/OS TASK STATES

A **task** is a separately executing thread with defined code, data, and stack. Several tasks may share the same code and possibly the same data, but they can never share the same stack. As Figure 1 shows, tasks in μC/OS may be in six states.

Figure 1—In the μC/OS kernel, all **task code is** loaded, but is dormant until created. The kernel runs the **highest priority** task that is ready. Running tasks can call delays or wait for an event (semaphore or queue). Interrupts may change the highest priority through the timer or interrupts. The scheduler then reassigns the highest priority status.



A **Dormant** task is in memory, has been linked, but is not currently assigned a priority or to a task control block (TCB).

A task is **Ready** after it has been created. It is assigned a TCP, a stack, a data area, and a priority.

A task is **Running** when the CPU is executing its code.

A task may be **Interrupted** when the CPU responds to an interrupt. The task may be suspended (returned to Ready with a context switch) if the interrupt changes the states of other, higher-priority tasks to Ready.

The task may be **Delayed** if it makes a delay for *N* ticks by calling `OSTa s kDe l a y ()`. After *N* ticks of the clock, the task is returned to the Ready and may run if it has the highest priority.

A task may be in a **Wait** state if it is waiting for a message, semaphore, or queue. Inherent in the design of μC/OS is a timeout for the Wait state. When a timeout is enabled, the task returns to the ready state with a return value that indicates that an error has occurred.

Tasks are made known to the μC/OS kernel with the `OSTa s kC r e a t e ()` call. They are deleted or put back into a Dormant state with the `OSTa s kDe l ()` call. Each task has a unique priority. Altogether, there are 64 priorities and a maximum of 63 tasks (The lowest priority task is preassigned to the `N U L L` task). Since tasks may change priority with the `OSTa s kC h a n g e P r i o ()` call, a full complement of 63 tasks would be inflexible.

A **Semaphore** is a signed integer which initializes to a positive integer or 0 before use. A positive value indicates the size of a resource while a negative value indicates how many tasks are waiting. Semaphores are created by `OS S e m C r e a t e ()`, which returns its Event pointer or "handle." A task that is waiting for the semaphore calls `OS S e m P e n d ()`. If the count value is positive, it decrements it and returns. If the value is 0 or negative, it decrements the counter and places the caller in a waiting list for the semaphore. It also may place the calling task in the wait state with a timeout value. A semaphore is signaled by calling `OS S e m P o s t ()`. The semaphore count is incremented. If the semaphore count is negative, then the waiting task with the highest priority is placed in the Ready state and its timeout value is zeroed. There is no delete semaphore call.

In μC/OS, a message is passed through a **Mailbox**, which is really a pointer value. Mailboxes are created through `OSM b o x C r e a t e ()` which returns an Event pointer. If a task wants the message, it calls `OSM b o x P e n d ()`. This call returns the message pointer and changes it to a `N U L L`. If the pointer is already `N U L L`, then the calling task is placed in Wait with a timeout. To post a message, a task calls `OSM b o x P o s t ()`. If the mailbox is

already full, it returns with an error. If there are tasks waiting on the mailbox, it sends it to the highest priority task, making it Ready, and resets the mailbox to `NULL`. There is no delete mailbox call.

Queues are similar to mailboxes, but they also allow for a definable set of items to be posted and used in FIFO fashion. With `OSQCreate()`, a task allocates an Event and an array of pointers to the Queue storage area. A task that desires an item from the Queue calls `OSQPend()`. If there are any available items, then the first one in the FIFO is popped off and returned. If the Queue is empty, then the caller is placed in Wait until there is something in the Queue or a timeout occurs. To place an item on the Queue, `OSQPost()` is called. If the Queue is full, then the call returns with an error. There is no delete Queue call.

1. ensuring the event vector you wish to use is disabled before changing it,
2. calculating the vector number you need, and
3. using `SWI_0x70` to install the vector.

Figure 4 gives a list of many of the DEMON system calls and Listing 2 offers a prototype of interrupt-vector installation in both assembler and C. In C-DEMON, the entire RDI/RDP protocol is handled by `main_prg()`,

which is reached after board initialization.

The RDI/RDP protocol operates in a half-duplex mode-one side is always waiting for the other. On reset, DEMON sends an "I'm alive" banner and waits for the host to respond with the program. When it gets to the forever loop in `main_prg`, the exchange is:

1. DEMON waits for a host command
2. DEMON interprets the command
3. DEMON goes back to step 1

This mode readily lends itself to the polled I/O method, which is how `DRIVER.C` is configured. I recommend the polled I/O method so you can debug interrupt-driven routines with DEMON configured for polled I/O. You can turn *all* interrupts off and still communicate with DEMON. So, if your program forgets to enable interrupts, you can examine the flags and see that this is so.

RETARGETING C-DEMON

Assuming your target's architecture is similar to the PID and that you do not need to rebuild the libraries provided by ARM (`armlib.321` or `armlib.32b`), there are still some constraints:

- RAM must be configurable to low memory (starting at `0x00000000`). This can be accomplished through direct physical addressing as in the PID or by use of the MMU virtual-to-physical remapping facilities.
- default starting location of a transient program is `0x00008000` (32 KB)

\$325

ARM Powered Single Board Computer

32 Bit ARM RISC with "FLAT" SVGA Video

Call for more information, a Programmer's Manual or Application Schematics

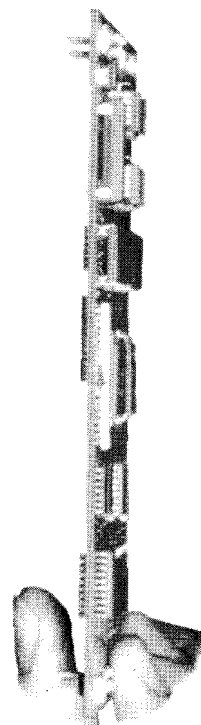
The Pixel Press video display processor is a complete video display subsystem in a 3 x 5 x .5 inch module. A flexible parallel interface allows connection to a Centronics Port, Parallel Port or directly to a Processor Bus. On board firmware can access external user hardware for embedded applications. Additional on board hardware includes a watch dog timer, voltage monitor, 4M bit EPROM, 256K Bytes Processor DRAM, 512K Bytes Frame Buffer and a Debug/Serial port. The debug port operates as either an RS-232 serial (TTL Level) port or supports direct connection of a PC-AT style keyboard. Various video output devices are supported including CRT (CGA, VGA & SVGA), EL and AM-LCD. With video timing provided via FPGA, other display modes are easily supported. Power requirements vary with display options. Typical power is 700ma at 5 Volts.

- | | |
|--|---|
| Resolution to 1024 x 768 Non-Interlaced | Library code available "Royalty Free" |
| Tools include C and Assembly | Application notes for Ethernet and SCSI |
| ROM support for many graphic primitives | Custom hardware & software assistance |
| Mounts to chassis or Printed Circuit Board | ROM based debug Monitor |

Anyone with experience in programming Intel 80x86 processors will find ARM assembly language a pleasant experience. With 14 general purpose registers and a flat memory address space programmers can manipulate 32 bit word, 8 bit bytes and pointers with extreme ease. No Selectors, No Segments, Just Plain Flat. A powerful barrel shifter is great for graphics. Conditional instructions and flag control keep jumps to a minimum and the processor pipeline full. A powerful Co-Processor interface can accelerate performance in custom hardware applications.

Applied Data Systems, Inc. - 409A East Preston St. - Baltimore MD USA

Tel: 1-410-576-0335 Fax: 1-410-576-0338 Toll Free: 1-800-541-2003



The header files `GLOBAL.H` and `DRIVER.H` have the equates and defines to modify your target. The file `DRIVER.C` needs a serial driver for the `GetByte` and `PutByte` routines. Get the serial routines working first. Although the timer is not needed at first, it will be missed by programs like `DHRY` (Dhrystone example program) or the μ C/OS real-time kernel, which tracks time. If you elect to add a driver file (such as an assembly-level driver), be sure to modify the `MAKEFILE` so that it is properly linked in.

Start with the simple stuff such as a new serial driver and `TimerINT()` (make it a `NULL` function). If you can put this in ROM and it works, great! Next, do the timer. When you have both the serial port and timer running, it's time for your additions to the DEMON. In this case, a two-step process of testing it in RAM and then ROM should reduce the development cycle.

The full source code for the C-DEMON is included in the standard software release from VLSI and is on the Circuit Cellar BBS. The rest is left to your imagination.

A MAJOR PORTING PROJECT

The μ C/OS real-time kernel was converted from 80186 to the ARM as an exercise to validate the ARM and its development environment under real-time constraints. This effort led to the radical rewriting of the DEMON.

μ C/OS is compiled separately and then linked into the user application to add real-time functionality. It does not need to be reinvented for each project. However, the current ARM μ C/OS implementation should be thought of as a work in progress. Those who are interested should read *μ C/OS: The Real-Time Kernel* and study both the 80186 and ARM code available from the Circuit Cellar BBS or VLSI.

μ C/OS supports a small range of basic elements necessary for the operation of a real-time environment: tasks, semaphores, messages, and queues. Tasks are defined in a preset number of *task-control blocks* (TCBs) and the other three have a common

Listing 3—The C code to start multitasking for the 80186 and ARM6 calls an assembly routine called `_OSStartHighRdy`, which loads the processor registers with those from the highest priority runnable task. Differences in the instruction construction of the two versions of this assembly routine are evident in the following examples.

```
void OSStart(void)

    UBYTE y, x, p;
    /* Find highest priority's task priority number */
    y = OSUnMapTbl[OSRdyGrp];
    x = OSUnMapTbl[OSRdyTbl[y]];
    p = (y << 3) + x;
    /* Point to highest prio task ready to run */
    OSTCBHighRdy = OSTCBPrioTbl[p];
    OSRunning = 1;
    OSStartHighRdy(); /* unload stack and start running */

/* _OSStartHighRdy 801861 version */
_OSStartHighRdy PROC FAR
    MOV AX, DGROUP
    MOV DS, AX
    MOV AX, WORD PTR DS:_OSTCBHighRdy+2
    MOV DX, WORD PTR DS:_OSTCBHighRdy
    MOV WORD PTR DS:_OSTCBCur+2, AX
    MOV WORD PTR DS:_OSTCBCur, DX
    LES BX, DWORD PTR DS:_OSTCBHighRdy
    MOV SP, ES:[BX]
    MOV SS, ES:[BX+2]
    POP DS
    POP ES
    POPA
    IRET
_OSStartHighRdy ENDP

void OSStartHighRdy(void)-ARM version
Start the task with the highest priority

OSStartHighRdy LDR a2,=OSTCBCur ;point at current context
                LDR a1,=OSTCBHighRdy ;TCB of highest prior.
                LDR a1,[a1] ;highest task ready to run
                STR a1,[a2] ;make it the current
                LDR a3,[a1] ;temp place sp in a3
; Start Next Context
                LDMIA a3!,{a1} ;get mode and PSR
                MSR CPSR,a1 ;restore the PSR (and mode)
                MOV sp,a3 ;put stack pointer in sp
                LDMIA sp!,{a1-ip,lr,pc};restore regs
```

Listing 4—This C code illustrates the use of `OS_ENTER_CRITICAL()` and `OS_EXIT_CRITICAL()` in μ C/OS. `OSTimeDly` delays the current task. If this code is moved to supervisor mode, then the calls to `OS_ENTER_CRITICAL()` and `OS_EXIT_CRITICAL()` may be eliminated.

```
void OSTimeDly(uint ticks)

    if (ticks > 0){
        OS_ENTER_CRITICAL(); /* Disable Interrupts */

        /* suspend the current task */
        if ((OSRdyTbl[OSTCBCur->OSTCBY] &= ~OSTCBCur->OSTCBBitX)==0)
            OSRdyGrp &= ~OSTCBCur->OSTCBBitY; /* then group not ready */

        OSTCBCur->OSTCBDly = ticks; /* load number of ticks in TCB */
        OS_EXIT_CRITICAL();
        OSSched(); /* find a new task to run */
```


data structure called an *event-control block* (Events). These also have a predefined number. Interrupts are special tasks initiated by the hardware and which may or may not interact with other tasks. The μ C/OS sidebar briefly describes μ C/OS features.

WHAT TO DO?

Like many programs of this type, μ C/OS is a mixture of assembler and C. Of course, the assembler portion needs to be reinvented. The C part of the code is also susceptible to change because of severe architectural differences between the parts. To port a program from the 8086 family, of which the 80186 is a member, the first thing to do is to compare some of the salient features of the two CPUs (see Table 2).

Data structures are the first to be converted since they define the details of the programs that must deal with them. Because of the difficulty the ARM has with 16-bit variables, it is often better to lengthen them to 32 bits rather than go through all the pain of jockeying packed 16-bit numbers.

ARM pointers are also converted to 32 bits (8086 far pointers have the same number of bits). The first data structure to be converted is the task-stack image. The 80186 task image was built to take advantage of a combination of **POP ES**, **POPA**, and **I RET** instructions. The ARM's task image uses the structure from a **Load Multiple Instruction** in which most of the registers are restored in one instruction.

The two important data structures—the task image and task control block—are offered in Tables 3 and 4.

TCB STRUCTURE

Because of the ARM preference for 32-bit quantities, the TCB is modified, even though 4-byte quantities could have been packed into a word. How these structures are used is best illustrated by the μ C/OS call **OSStart()**, which starts the multitasking kernel by starting the highest-priority task. Listing 3 gives the C code followed by the 80186 and ARM assembler support codes.

Table 2-A rough comparison of the 80186 and ARM architectures shows only minor overlaps.

Item	80186	ARM
Word Size	16/8 bits	32/8 bits
Address Range	64-KB off. + I-MB seg.	4-GB linear
Number of Regs	8 + 4 segment regs	16 + 15 overlapping regs
Supervisor Modes	None	FIQ, IRQ, Abort
Instruction Size	1-7 bytes	4 bytes

Description	80186 (large mem)	size	ARM size	Description			
Data Area	{	Offset of Data	16	R15-PC	32 = Code Pointer		
		Segment of Data	16	R14-LR	32		
Code Area	{	offset of Code	16	R12	32		
		Segment of Code	16	R11	32		
Status Word 8	{	PSW	16	R10	32		
CS:IP Restored by IRET	{	IP (PC)	16	R9	32		
		c s	16	R8	32		
Registers are restored by POPA	{	Ax	16	R7	32		
		CX	16	R6	32		
		DX	16	R5	32		
		BX	16	R4	32		
		SP	16	R3	32		
		BP	16	R2	32		
		SI	16	R1	32		
		DI	16	R0	32 = Data Pointer		
		POP ES	{	ES <- stk ptr	16	PSR	32
						R13	32 <- stk ptr
Total		16 items		17 items			

R0-R15 restored by LDMFD SPI, {R0-R12, LR, PC}

Table 3-Comparing the stored processor state for the 80186 and the ARM6 shows the number of items to be about the same. However, the ARM6 registers are 32 bits wide versus the 80186's 76 bits.

Listing 5-The 9 est.c program is a good example of programming with DEMON and μ C/OS.

```
int main (void)
{
    ret3parm retval;
    int id[NUM_TASKS], j;

    union gp {
        unsigned char *b;
        unsigned int *w;
    } p;
    p.b = (unsigned char *) IOBase;

    for (j=0; j<NUM_TASKS; j++) /* generate the IDs*/
        id[j] = (int)'1' + j; /* create an ID we can see */

    OSInit(); /* needed by uC/OS */
    DispSem = OSSemCreate(1); /* Display semaphore */
    pQcb = OSQCreate((void **)Qmsg, QDEPTH); /* queue */
    j=0;
    OSTaskCreate(TaskEmptyFill, &id[j], &TaskStk[j][TASK_STK_SIZE], j+1);
    j++;
    OSTaskCreate(TaskEmptyFill, &id[j], &TaskStk[j][TASK_STK_SIZE], j+1);
    j++;
    OSTaskCreate(TaskEmptyFill, &id[j], &TaskStk[j][TASK_STK_SIZE], j+1);
    j++;
    OSTaskCreate(TaskFill, &id[j], &TaskStk[j][TASK_STK_SIZE], j+1);
    j++;
    OSTaskCreate(IdleTask, &pZERO, &TaskStk[j][TASK_STK_SIZE], j+1);

    SWI_Write0("\nHooking into the C-DEMON's PANIC button...\0");

    /* Now HOOK the DEMON's PANIC button to uC/OS */
    /* PANIC is bit position 7 in IRQ
    (vecnum = IRQbitvector+bit1 = 0x20 + 7)*/
    retval = SWI_InstallHandler(0x27, 0, Panic_IRQ);
    if (retval.p2 == NULL)
        SWI_Write0("\nError: Unable to install PANIC handler.\n");
    if (retval.p1 != 0x27)
        SWI_Write0("\nError: vecnum is NOT 0x27.\n");
}
```

(continued)

Although the simple ARM instructions make code easier to read, the fact that the ARM does have processor modes makes some μ C/OS functions more complicated. Whenever a μ C/OS function requires access to private data structures that must be completed without interruption, the C code calls `OS_ENTER_CRITICAL()`. After the sensitive code is finished, `OS_EXIT_CRITICAL` is called.

In the 80186, these are simply translated to `CLI` (clear interrupt enable) and `STI` (set interrupt enable). Although in user mode the ARM cannot change the interrupt enables directly, it can through an operating system call. Thus, `SWI 0x14` (disable interrupts) and `SWI 0x13` (enable interrupts) are used.

The process of going through the `SWI` call takes many cycles, as the ARM must save user registers, back up R14 to find the `SWI` code, run through a lookup table, and then do the request. Many μ C/OS functions call `OS_ENTER_CRITICAL()` and `OS_EXIT_CRITICAL()` and place them around their user code to accomplish the function. Listing 4 offers an example of delaying a task. Thus, the processor may go through a mode transition up to four times for each call.

The next step in porting μ C/OS involves a major rewrite. All OS functions will be placed in Supervisor Mode and operated from `SWI` calls. The interrupt enable (for IRQ) will be automatically turned off after a `SWI` instruction, obviating the need for the `OS_ENTER-` and `OS_EXIT_CRITICAL` functions.

PUTTING IT ALL TOGETHER

Included in the BBS distribution for ARM μ C/OS is a program called `qt.e.s.t.c`, part of which is in Listing 5. This program gives a good example of programming with the DEMON API as well as the μ C/OS kernel. In addition to producing a colorful display on the host's screen, `Qt.e.s.t` creates four tasks and an idle task as well as two interrupt service routines.

In Listing 5, `main()` gives an overview of the task-initialization process. The program creates a semaphore with

Listing 5—continued

```

/* Now HOOK the DEMON's TIMER to uC/OS */
/*(p.b + IRQM) = 0: /* stop the TIMER interrupt */
/* TIMER is bit position 1 in IRQ
   (vecnum = IRQbitvector+bit1 = 0x20 + 1) */
retval = SWI_InstallHandler(0x21, 0, Timer_IRQ);
if (retval.p2 == NULL)
    SWI_Write0("\nError: Unable to install handler.\n");
if (retval.p1 != 0x21)
    SWI_Write0("\nError: vecnum is NOT 0x21.\n");
prev_hand = retval.p3;
*(p.b + IRQM) = TimerIT; /* restart the TIMER interrupt */
SWI_Write0("\nDoing an OSStart()\n\n\0");
OSStart(); /* start the pandemonium */
}

/*Panic_IRQ * When the PANIC button is pushed, issue a message*/
void Panic_IRQ(void)
{
    union gp {
        unsigned char *b;
        unsigned int *w;
    } p;
    p.b = (unsigned char *)IOBase;
    *(p.b + IRQRST) = Panic; /* reset any PANIC interrupt */
    SWI_Write0(CYAN);
    SWI_Write0("***OUCH***\0");
}

```


(continued)

To Find the Best TMS320 DSP Development Tools Come to the Mountains

Mountain-30
 TMS320C3x Emulator/Target System
 Full-Featured PC-AT C3x Emulator with
 On-Board TMS320C30 DSP, Starting at \$3995

Mountain-5 10
 Universal Emulator
 Complete PC-AT-Based Emulation for the TMS320C3x,
 TMS320C4x, and TMS320C5x Families, Starting at \$3495

Mountain-40
 TMS320C4x
 Evaluation Module
 Single TMS320C40/Half-Size
 PC-AT Card, Expansion Galore,
 C Source Debugger Included,
 Starting at \$3995


 WHITE MOUNTAIN DSP
 131 DW Highway, Suite 433
 Phone (603) 883-2430

Listing 5—continued

```

/* Timer_IRQ * Timer interrupt routine */
void Timer_IRQ(void)
{
    union gp {    unsigned char *b;
                  unsigned int *w;
                } p;
    p.b = (unsigned char *)IOBase;
    *(p.b + IRQRST) = TimerIT; /* reset Timer interrupt */

    SaveCtx();          /* Save the USER context */
    OSTimeTick();       /* do uC/OS tick routine */
    RestoreCtx();       /* Restore the USER context */
}

```

a count of 1 and a queue with a depth of 32 items. It then creates five tasks, three of which have the same code and are called `TaskEmptyFill`. It also creates `TaskFill` and `IdleTask`. `TaskEmptyFill` alternately empties and fills the queue, while `TaskFill` only fills it. The idle task prints dots on the screen. The program connects the PID Panic button to the Panic ISR using the `SWI_InstallHandler` system call. It then hooks up to the μ C/OS Timer ISR with the same call. The last call before pandemonium breaks out is the `PC/OSStartO`. I hope your code is more useful!

INSTRUCTION SET EMULATION

The previous example of converting μ C/OS was aided by the availability of the source code. Converting a piece of code originally written for another processor when we have the source code can be very tedious. It is possible to do some of this more automatically

by writing macros that directly translate code into ARM assembler.

With the 68000, Marco Graziano did just that and converted the sieve (of Eratosthenes) program into ARM code. Even though the 68k registers were kept in memory, the PID was able to beat a 68020 in this benchmark. The use of macros, however, causes massive code growth. Besides, often the source code is unavailable, so you only have the binary machine code.

To solve this problem, the old code can run on a machine-code emulator. Such a tactic is used by Acorn to run 80x86 PC code. A similar program called `SoftPC` runs 80x86 code on a Macintosh or Sun.

FUTURE ARM DEVELOPMENTS

Nothing in the IC and electronics business is static, especially in the world of VLSI (generic) and RISC processors. The ARM-based product

line will be enhanced with faster and more capable processors as well as whole systems on a chip. When these are mature, I would be happy to inform you as readers of *Circuit Cellar INK* about the products and how you can use them for your own projects.

In the meantime though, you can get started on the ARM processor using the cross-development toolkit for building ARM-based projects. With this toolkit, you can write, link, and debug code including C and assembly. Large software projects, including a port of UNIX, have been developed with this environment. I have also presented the DEMON board-level debugger and the small, but useful μ C/OS real-time kernel. I trust this material helps you make progress on your ARM-based projects. □

I would like to thank the software tool developers at ARM Ltd., especially Marco Graziano, Geary Chopoff, and Jaime Smith, for their help with the software used in this article.

Art Sobel is the hardware applications manager for embedded products at VLSI Technology. He has spent 24 years in Silicon Valley designing disk drive electronics, disk drive controllers, laser interferometers, laser printer controllers, many controller chips, and speech synthesizers. He can be reached at sobel_a@vlsi.com.

REFERENCES

van Someren, Alex, and Carol Atack, *The ARM RISC Chip: A Programmer's Reference Manual*. Addison-Wesley (1993), ISBN 0-201-40695-0.

Labrosse, Jean L. *μ C/OS: The Real-Time Kernel*. R & D Publications, (1992), ISBN 0-13-031352-1.

SOFTWARE

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

Item	80186	size	ARM	size	Description
OSTCBStkPtr	far *	16+16	Pointer	32	Pointer-to-task stack image
OSTCBStat	ubyte	8	uint	32	Task status
OSTCBPrio	ubyte	8	uint	32	Task priority
OSTCBDly	uword	16	uint	32	Timeout for delay or wait
OSTCBX	ubyte	8	uint	32	Priority byte bit position
O S T C B Y	ubyte	8	uint	32	Priority group bit position
OSTCBBitX	ubyte	8	uint	32	Precalculated bit mask
OSTCBBitY	ubyte	8	uint	32	Precalculated bit mask
● OSTCBEventPtr	Pointer	16	Pointer	32	Pointer to Event Control Block
*OSTCBNext	Pointer	16	Pointer	32	Pointer to next TCB
*OSTCBPrev	Pointer	16	Pointer	32	Pointer to previous TCB

*OSTCBStkPts points to the bottom of either task structure

Table 4—The ARM6 version of the μ C/OS task-control-block (TCB) structure has a larger item size due to the larger word size in the ARM6.

SOURCES

VLSI Technology
8375 River Pkwy.
Tempe, AZ 85284
(602) 753-6373
Fax: (602) 753-6001
tom.schild@tempe.vlsi.com

Other suppliers of ARM processors, software, PIE boards, and information:

GEC Plessey Semiconductors
1500 Green Hills Rd.
Scotts Valley, CA 95066
(408) 4382900
Fax: (408) 438-5576

Cheney Manor
Swindon
Wiltshire
United Kingdom SN2 2QW
(0793) 518-000
Fax: (0793) 518-411

Sharp Microelectronics
5700 NW Pacific Rim Blvd.
Camas, WA 98607
(206) 834-2500

Other ARM board suppliers:
Applied Data Systems, Inc.
409A East Preston St.
Baltimore MD 21202
(410) 576-0335
Fax: (410) 576-0338

ARM software suppliers:
Perihelion Distributed Software
The Maltings, Shepton Mallet
Somerset, UK BA4 5QE
(0749) 344-345
Fax: (0749) 344-977
pds@perihelion.co.uk

RISC-OS
Acorn Computers Ltd.
Acorn House
Vision Park, Histon
Cambridge, UK CB4 4AE
(0223) 254-222
Fax: (0223) 254-262
customer.services@acorn.co.uk

IRS

410 Very Useful
411 Moderately Useful
412 Not Useful

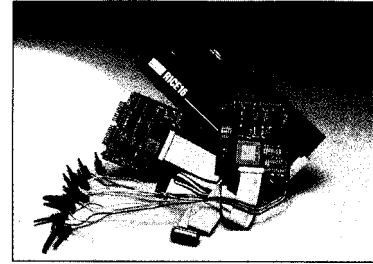
PIC16C5x/16Cxx Real-time Emulators

Introducing RICE16 and RICExx-Juniors, real-time in-circuit emulators for the PIC16C5x and PIC16Cxx family microcontrollers: affordable, feature-filled development systems from **\$599***

*Suggested Retail for U.S. only

RICE16 Features:

- Real-time Emulation to 20MHz for 16C5x and 10MHz for 16Cxx
- PC-Hosted via Parallel Port
- Support all oscillator types
- 8K Program Memory
- 8K by 24-bit real-time Trace Buffer
- Source Level Debugging
- Unlimited Breakpoints
- External Trigger Break with either "AND/OR" with Breakpoints
- Trigger Outputs on any Address Range
- 12 External Logic Probes
- User-Selectable Internal Clock from 40 frequencies or External Clock
- Single Step, Multiple Step, To Cursor, Step over Call, Return to Caller, etc.
- On-line Assembler for patch instruction
- Easy-to-use windowed software
- Support 16C71, 16C84 and 16C64 with Optional Probe Cards
- Comes Complete with TASM16 Macro Assembler, Emulation Software, Power Adapter, Parallel Adapter Cable and User's Guide
- 30-day Money Back Guarantee
- Made in the U.S.A.



Emulators for **16C71/84/64** available now!

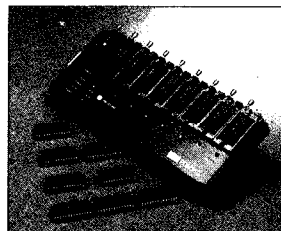
RICE-xx Junior series

RICE-xx "Junior" series emulators support; PIC16C5x family, PIC16C71, PIC16C84 or PIC16C64. They offer the same real-time features of RICE16 with the respective probe cards less real-time trace capture. Price starts at \$599.

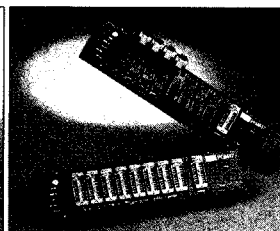
PIC Gang Programmers

Advanced Transdata Corp. also offers PRODUCTION QUALITY gang programmers for the different PIC microcontrollers.

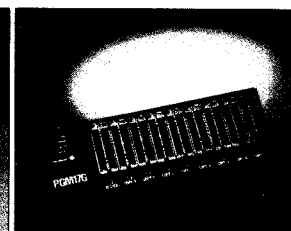
- Stand-alone COW mode from a master device
- PC-hosted mode for single unit programming
- High throughput
- Checksum verification on master device
- Code protection
- Verify at 4.5V and 5.5V
- Each program cycle includes blank check, program and verify eight devices
- Price starts at **\$599**



PGM16G: for 16C5x family



PGM47: for 16C71/84



PGM17G: for 17C42

Call **(214) 980-2960** today for our new catalog.

For RICE16.ZIP and other product demos, call our BBS at (214) 980-0067.



Advanced Transdata Corporation Tel (214) **980-2960**
14330 Midway Road, Suite 120. Dallas, Texas 75244 Fax (214) 980-2937

DEPARTMENTS

42

Firmware Furnace

50

From the Bench

100

Silicon Update

108

Embedded Techniques

115

ConnecTime

FIRMWARE FURNACE

Ed Nisley

Journey to the Protected Land: Serious CISC Meets the Taskettes



This month, Ed introduces us to fundamental multitasking using the '386's features. And, once he has that going right, he takes out his stop watch to measure just how long the task switch takes.



Contrary to what RISC proponents may claim, the Intel 80386 is not the avatar of CISC architectural complexity. Baroque, yes; barnacle-encrusted, yes; the most intricate, no. My vote goes to a certain (mercifully canceled) main-frame that sported, among other oddities, a PAA instruction-Perform Alternate Architecture. Now *that* was a complex instruction!

This month we'll pop the top on multitasking, arguably the most intricate area of this unabashedly CISC CPU. The '386SX leaves us little choice because desirable features such as Virtual 86 mode, paged memory, and exception handling depend on tasks. Exploring the most prominent peaks of this region will take several months even with crampons.

The code this month includes `StrFormat`, `asprintf0` which produces formatted output. You may find it helpful as a lightweight numeric converter in applications that don't need full-bore ANSI compliance.

THE SIGHT OF TWO TASKS SWAPPING

Simply put, a task is what the CPU does when it's running a pro-

Listing 1—After initializing the hardware and software, the FFTS code enters an idle loop, now grandly called the Kernel (hey, it's a start). On each iteration, the code updates and displays a loop counter, pulses a parallel port bit, and calls the dispatcher routine to execute a task switch.

```

@@KernelIdle:
    MOV  EAX,[StatusCtr] ; use high word of count
    SHR  EAX,16
    CALL UtilByteToLEDS,EAX
    INC  [StatusCtr] ; ready for next iteration

    MOV  EDX,SYNC_ADDR ; send a blip
    IN   AL,DX
    OR   AL,01h
    OUT  DX,AL
    AND  AL,NOT 01h
    OUT  DX,AL
    CALL TaskDispatch ; do the task switch
    JMP  @@KernelIdle ; and repeat forever!

```

gram. Multitasking is just switching from one program to another, preserving the state of the first program, and then loading the second. Switching rapidly enough between programs gives the illusion of making progress everywhere at once. The sham is successful only because the CPU is quicker than the eye.

In the 80386 architecture, a program's entire state resides in a task state segment (TSS) when the CPU is not running it. The TSS holds the general and segment registers, current instruction address, stack location, and other familiar values. There are also, as we will see, a few unfamiliar items.

Each TSS, being a segment, must have a descriptor in the global descriptor table (GDT); the selector corresponding to the GDT entry uniquely identifies the task. The CPU's task register (TR) holds the TSS selector of the current task. During a task switch,

the CPU stores the program state in the TSS pointed to by the TR.

Rather than deploying a real-time multitasking kernel, I'll start off with the minimum-two trivial tasks that swap control back and forth. Bitasking taskettes require much of the same setup and overhead as multitasking big tasks while omitting the complexity that obscures essential details.

Listing 1 shows the first taskette: the same FFTS idle loop, familiar from previous columns, is now grandly called the Kernel task. This endless loop updates a counter, blips a parallel port bit, and calls the task dispatcher to switch to the other taskette before branching back to its start. Prior to this loop, the FFTS code performs the start-up functions described last month, initializes the hardware, and prepares a TSS for each taskette.

The other taskette, called Demo Task in Listing 2, loops endlessly

while blipping a different parallel port bit. It calls the same task dispatcher function to return control to the kernel taskette. A pulsing port bit is the only indication we have that this taskette is running.

Listing 3 presents a complete, albeit stripped-down, task dispatcher for the taskettes. ThisTaskPtr is a 48-bit FAR pointer holding the TSS selector of the current task. Next TaskPtr holds the selector of the next task to be executed. Obviously, with only two tasks, it's also the selector for the previous task.

If 48-bit pointers seem excessive, bear in mind that they're just the 32-bit, protected-mode equivalent of real-mode FAR pointers. Sixteen of those bits hold the PM segment selector, which must be a TSS in the GDT. The remaining 32 bits are an offset within a segment that may span 4 GB. In this case, strangely enough, the offset will always be zero because the CPU gets the actual branch target from the TSS.

TaskDispatch swaps the segment selector portions of the two pointers, sets a parallel port bit, then executes an indirect JMP through ThisTaskPtr. The task switch occurs during this single instruction, saving the current CPU state in the outgoing TSS and loading the new state from the incoming TSS. The first few instructions after the jump in the new task turn the port bit off and return.

The scope traces in Photo 1 show those three chunks of code at work. The two taskettes produce the pulses in the top two traces. The bottom trace is the task dispatcher's output. That 16- μ s pulse marked by the timing cursors is the indirect JMP doing the task switch!

It bears emphasis: the JMP instruction marked by those pulses is the task switch. The CPU executes one instruction with one explicit memory operand, stores dozens of bytes in one TSS, reads a similar block from another TSS, while loading and validating all the segment selectors, memory references, TSS contents, and so forth and so on. The JMP occurs in one task and the next instruction is in another.

Serious CISC, indeed!

Listing 2—This task gains control whenever the FFTS kernel does a task switch. It also pulses a (different) parallel port bit and calls the task dispatcher. The dispatcher preserves the caller's registers, eliminating the need to reload EDX in the loop.

```

PROC DemoTask

MOV  EDX,SYNC_ADDR ; this is preserved forever
@@Again:
    IN   AL,DX ; send a blip
    OR   AL,02h
    OUT  DX,AL
    AND  AL,NOT 02h
    OUT  DX,AL
    CALL TaskDispatch ; do the task switch
    JMP  @@Again ; and repeat
ENDP DemoTask

```

THE UNITED STATES OF TASKING

The setup for those single-instruction task switches requires considerably more effort than executing them. The TSSs and their descriptors must coordinate correctly with each other and their own code, data, and stack segments. In the general case, getting this right can be a nightmarishly complex, ummm, task.

Figure 1 shows the simplified storage layout we'll use for the next few months. The TSS descriptors begin at GDT_TSS_BASE in the GDT. Each TSS descriptor is followed by the task's LDT descriptor, although we don't need or use LDTs this month.

A TSS descriptor specifies a task state segment, allowing the CPU to perform task switches into and out of that task. Attempting to load a TSS descriptor into any CPU segment register other than the TR causes an immediate protection exception. You cannot read or write a TSS using its descriptor, even though the descriptor includes the segment's starting address and length. You must initialize TSS fields through a separate data segment descriptor.

The general solution requires a unique segment descriptor called a *data alias* for each TSS. That descriptor gives you read and write access to the TSS, and when you're done, you discard the alias. I took a slightly

different approach by arranging all the TSSs in an array starting at address 00130000 covered by a single data descriptor called GDT_TSS_ALIAS. The task-creation code converts each task selector into an array index, then aims

ES : EDI at the start of the corresponding TSS. An assembly language ST RUC gives easy access to the fields within each TSS.

Listing 4 presents the definition of those TSS fields. There are three major sections: the machine state between offset 0 and IOMapBase, an optional data area, and the I/O permission bitmap at the end of the segment. Because the machine state is the only required part, the smallest possible TSS is a mere 68h (104 decimal) bytes long.

Many of the two-byte fields, such as the segment registers, are padded with two bytes of binary zeros to preserve double-word alignment. While it is tempting to fit user data into these niches, the Intel doc specifically reserves them by mandating zero fill. Disturb not the reserved areas!

For our present purpose, the essential part of the machine state begins with EIP and ends with the GS field. The BackLink, StackPtr, CR3,

Listing 3-An indirect JMP instruction performs a '386 task switch when the memory location holding the target address has a task's JSS selector. The JMP target in this code alternates between the two TSS selectors corresponding to the two taskettes. The B L register is restored from the incoming task's JSS and will change even though it's not explicitly reloaded!

```

PROC TaskDispatch
USES EAX,EBX,EDX

swap the task pointers
MOV AX,[ThisTaskPtr.Seg]
XCHG AX,[NextTaskPtr.Seg]
MOV [ThisTaskPtr.Seg],AX

- do the task switch
STR BX ; get current task register
MOV EDX,SYNC_ADDR2 ; show it on LPTP
MOV AL,BL
OUT DX,AL
MOV EDX,SYNC_ADDR ; mark the start in the old task
IN AL,DX
OR AL,04h
OUT DX,AL

JMP [FWORD PTR ThisTaskPtr] ; shazam!

IN AL,DX ; mark the end in the new task
AND AL,NOT 04h
OUT DX,AL
MOV EDX,SYNC_ADDR2 ; show new task
MOV AL,BL ; . . . BL restored from the TSS
OUT DX,AL

return to the new task
RET
ENDP TaskDispatch

```

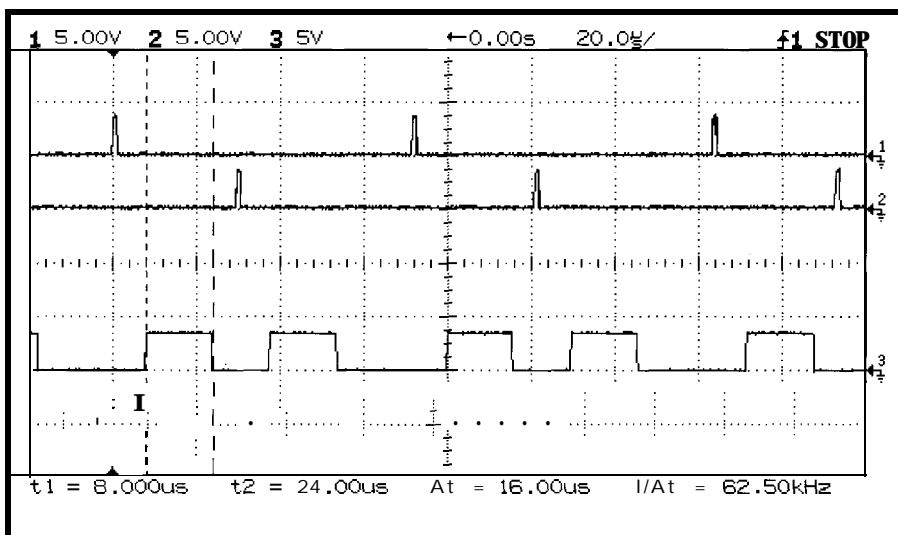


Photo 1 —The indirect JMP performing a '386 protected-mode task switch is unlike any JMP you've seen before. The two tasks produce the pulses in the first two traces. The task dispatcher routine sets the bottom trace high just before the task-switching JMP and low immediately afterward. The pulse is 16 μ s long; the JMP itself requires about 15 μ s or 500 clock cycles at 33 MHz!

LDTsel, **TrapEnable**, and **IOMapBase** fields aren't needed for our taskettes. They remain present, however, and must be zero-filled to prevent the CPU from acting on them, as there is no way to do a partial task switch.

The optional data area in our TSS structure holds two items. A 32-byte character string identifies the task in readable ASCII for use by the TSS dump routine. The task's local descriptor table (LDT) has room for **16** descriptors, although it simply soaks up space this month.

The I/O permission bitmap must begin within 64 KB at the start of the TSS because the **IOMapBase** field is only **16** bits long. The ISA bus I/O address space has 1024 ports, each corresponding to a single map bit. Our bitmap thus occupies 128 bytes and, with the LDT, simply soaks up space until we need it in a few months.

The code in Listing 5 sets up the TSS descriptor and fills the key TSS fields for the **DemoTask** function shown in Listing 2. The descriptor must contain the TSS's linear base

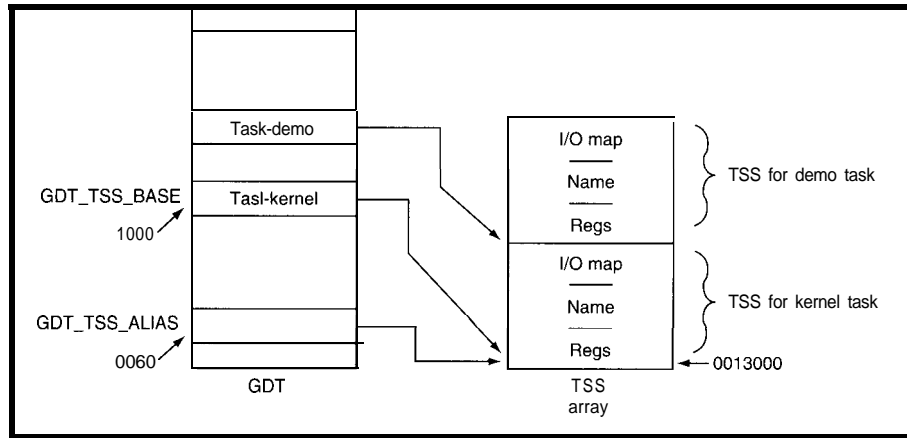


Figure 1—FFTS arranges the task state segments (TSS) in an array starting at address 00130000. The corresponding descriptors in the GDT begin at **GDT_TSS_BASE** (selector 1000) and occupy every other descriptor entry. In later columns, each task's LDT descriptor will follow its TSS descriptor. The **GDT_TSS_ALIAS** descriptor (selector 0060) provides read-write data access to the TSS array.

address rather than its offset within the **GDT_TSS_ALIAS** segment. The starting **CS: EIP** values are simply the **GDT_CODE** segment and the offset of **DemoTask's** first instruction.

The **SS: ESP** fields must point to an area large enough to hold **DemoTask's** stack. Rather than define a completely new stack segment, I split the (overly large) existing stack in half

and set **SS: ESP** to the top of the lower half. That division allocates about 28 KB of stack to each task and, with interrupts disabled, gives new meaning to the old saw "Nothing exceeds like excess."

The remaining segment-register fields contain the same values as the **KernelTaskette**. **DemoTask** and **Kernel** can share segments because

Protected
Mode
Stress?



Ease into Protected Mode
with *pmEasy*™

pmEasy is a complete protected mode environment for embedded systems. It initiates protected mode and provides an application loader, trap handler, error handler, memory manager, debugger support, screen writes and more. *pmEasy* is integrated with low-cost 16- and 32-bit development tools from Microsoft, Borland, Periscope, and others.

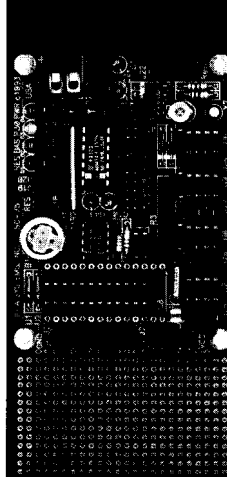
Why struggle developing your own protected mode environment? *pmEasy* lets you focus on your application.

pmEasy16 or 32 **\$495** including source code

BUY AND TRY 30 day money-back guarantee

MICRO DIGITAL INC **1-800-366-2491**
Developer of *smx*
Cypress, CA, USA FAX 714-691-2363 VISA, MC, AMEX

8051
80C32



87C751
87C752

Use one of our embedded controllers to save time and money. They are ideal for developing products, test fixtures and prototypes.

We offer a complete line of controller boards and software tools for the 8051 and 87C751 families of microcontrollers. Complete packages are available to help you develop your projects.

Features:

- Breadboard area
- Flexible I/O arrangement
- Powerful controller BASIC for the 87C752 or 80C32

Ph: (702) 83 1-6302

Fax: (702) 83 1-4629

Iota Systems, Inc.
POB 8987 • Incline Village, NV
89452-8987

they're harmless. The casual approach suffices for this month's taskettes and fails miserably in the general case. Next month, we'll install fire walls between the tasks at the cost of considerably more setup code.

Figure 2 displays the contents of both TSSs before the first task switch. All the `Kernel` TSS fields are zero because the CPU stores its current state during the first task switch. Only the essential `DemoTask` TSS fields are nonzero. You can see the value of the name field to identify those otherwise anonymous hexits!

TIME ENOUGH FOR TASKING

According to the data book, a '386SX indirect `JMP` task switch requires 328 cycles or 10 μ s at 33 MHz. The 16- μ s pulses shown in Photo 1 include 1 μ s to create the output pulse and 15 μ s for the task switch itself. That's about 500 CPU cycles.

An experiment with my system's CMOS configuration settings sheds some light on those 170 extra cycles. One additional read wait state adds 60 cycles, one additional write wait state adds 17, and both together add 82. Given the resolution of reading a scope, if the system board imposes a few wait states even when set for "0 W/S," the mystery is solved.

Can this be so! Beats me! As with most clones, the exact function of the BIOS setup options isn't particularly well documented.

In any event, those additional cycles indicate that the CPU makes about 80 memory accesses during each task switch. That's in rough agreement with the number of registers and values in Listing 4. The CPU must store the current task state in one TSS before reading the new task state from another. Underneath that activity, the '386SX bus interface unit cracks each 32-bit access into two 16-bit bus transactions. The memory gets more exercise than may be evident at first glance.

The CPU saves all the registers regardless of whether the tasks actually use them. Even though we don't have full intertask fire walls in place, you can see how the CPU can

Listing 4—This assembler structure defines the layout of a task state segment (TSS). The '386SX CPU stores its registers in the current task's JSS during a task switch, then loads them from the new task's TSS. When a task isn't active, the TSS contains all of the CPU state required to resume it. The data between `IOMapBase` and `IOMap` is optional; `FFTS` stores the task's name and the LDT in that spot.

```

                STRUC    TSS
;-- CPU-defined fields
BackLink    DW    ?,0    ; previous TSS selector
StackPtr0   FULLPTR {}   ; SS:ESP for CPL 0
                DW    0
StackPtr1   FULLPTR {}   ; SS:ESP for CPL 1
                DW    0
StackPtr2   FULLPTR {}   ; SS:ESP for CPL 2
                DW    0
CR3         DD    ?      ; paging setup
EIP         DD    ?
EFLAGS     DD    ?
EAX        DD    ?
ECX        DD    ?
EDX        DD    ?
EBX        DD    ?
ESP        DD    ?
EBP        DD    ?
ESI        DD    ?
EDI        DD    ?
ES         DW    ?,0
cs         DW    ?,0
SS         DW    ?,0
DS         DW    ?,0
FS         DW    ?,0
GS         DW    ?,0
LDTsel     DW    ?,0    ; LDT selector
TrapEnable DW    ?      ; 0=not, 1 trap on start
IOMapBase  DW    ?      ; OFFSET IOMap if used, else 0

;-- custom fields for FFTS task management
                the CPU knows nothing of these, so they're not
                changed automagically

TASKNAME_SIZE =    31    ; longest possible name

TaskName    DB    TASKNAME_SIZE DUP (?) ; room for the string
                DB    0      ; ensure a terminator

;-- LDT for this task's private code and data

TASKLDT_SIZE =    16    ; number of LDT entries

TaskLDT     DD    TASKLDT_SIZE DUP (?) ; 8-byte slots

;-- I/O permission bitmap, 0 = enabled, 1 = disabled
                we only cover the first 1024 ports used in ISA bus systems

IOMap       DB    128 DUP (?) ; all I/O ports, 8 per byte
IOMapEnd    DB    ?      ; must be FF

                ALIGN    4      ; put next value on DWORD boundary
                ENDS    TSS

```

enforce considerable protection when it's needed. The downside is that you get all of the overhead regardless of how little isolation you actually need.

RISC proponents point out that task switching need not be so complex and, instead, should use a sequence of simple, fast, cheap instructions. Even

the Intel manuals suggest some systems can use the overall '386SX TSS layout with a subroutine that saves only a small subset of the full machine state. Of course, you can't use both approaches in the same system without considerable forethought because an incomplete or

invalid TSS will cause a protection exception.

The CISC approach is faster than the exact same operations carried out by a subroutine because the CPU fetches and decodes only a single instruction. The RISC technique is faster if you save fewer registers and perform fewer protection checks. If you need a balance point somewhere between those extremes for your system, fire up your scope and logic analyzer. For obvious reasons, I will use the full-bore '386SX approach for FFTS.

A particular problem with CISC task switching occurs in embedded-control systems requiring very fast interrupt response. Because the task switch is one *loonnng* uninterruptable instruction, there may be 15 μ s or more before the CPU can respond to an IRQ. Practical operating systems wrap additional uninterruptable code around the switch, which means the actual delay may depend more on the code than the CPU hardware.

Now that you have the Big Picture, let's take a closer look at the events surrounding a single task switch.

MAKING THE SWITCH

The `TaskDispatch` code in Listing 3 looks just like an ordinary function. The `US ES` directive generates hidden code to save the registers on the stack when the routine gets control and restores them before it executes the `RET` instruction. All this is quite standard, save for one fact: the stack at the end of the routine isn't the same as the stack at the beginning.

Or is it?

When `Kerne1`'s idle loop calls `TaskDispatch` for the first time, the `CALL` instruction uses the stack defined by the startup code. The saved registers and return address appear near the top of the stack segment between 00122000 and 0012FFFF. The stack is ready for a normal return, but that's not what happens.

`TaskDispatch` swaps `ThisTaskPtr` and `NextTaskPtr`, placing `DemoTask`'s TSS selector in `ThisTaskPtr`. It `JMP`s indirectly through that pointer causing a task switch. The

Listing 5—*This code creates a TSS descriptor with the `TASK_DEMO` selector and a TSS for the second taskette. The TSS descriptors begin at `GDT_TSS_BASE` in the GDT and the segments themselves are arrayed starting at `GDT_TSS_ALIAS`. For simplicity, the code, data, and constant segments are shared between the two taskettes, and `SS:ESP` points to an unused part of the original FFTS stack segment. The `StrNCopy` function copies a name string into the TSS's name field where it identifies the task.*

```
TSS_PTR EQU <<(TSS_PTR ES:EDI)>> ; shorthand notation
MOV EDI,((TASK_DEMO-GDT_TSS_BASE)/TSS_DESCSTEP) * \
TSS_SPACING ; offset addr of TSS in data

CALL MemGetDescBase,GDT_TSS_ALIAS,GDT_GDT_ALIAS
ADD EAX,EDI ; linear + offset in segment
MOV EDX,EAX ; EDX = linear address of TSS
CALL MemSetDescriptor,TASK_DEMO,GDT_GDT_ALIAS, \
EDX, SIZE TSS, ACC_TASK32, 0

LEA EAX,[TSS_PTR.TaskName]; get string addr
CALL StrNCopy,GDT_TSS_ALIAS,EAX,TASKNAME_SIZE, \
GDT_CONST,OFFSET DemoName

MOV [TSS_PTR.CS],GDT_CODE ; aim at start of task
MOV [TSS_PTR.EIP],OFFSET DemoTask

MOV EAX,GDT_STACK ; split the stack area in half
MOV [TSS_PTR.SS],AX
LSL EAX,EAX
SHR EAX,1
SUB EAX,3 ; leave top dword alone
MOV [TSS_PTR.ESP],EAX
MOV [TSS_PTR.DS],GDT_DATA ; set up remaining segs
MOV [TSS_PTR.FS],GDT_CONST
CALL TaskDumpTSS,TASK_DEMO
```

current CPU state includes `SS:ESP`, locating the stack, and `CS:EIP`, identifying the next instruction. In this case, even though the instruction was a branch, the CPU stores the address of the instruction immediately after the `JMP`, not the actual indirect target address, in `Kerne1`'s TSS.

The CPU registers now fill from `DemoTask`'s TSS. As you saw in Listing 5, `DemoTask`'s stack occupies the lower part of the stack segment at about 00129000. That stack has nothing in it yet, so a return would be disastrous.

The `DemoTask` TSS also supplies the address of the first instruction the CPU should execute after the task switch. The `CS:EIP` fields stored in Listing 5 aim the CPU at the start of the `DemoTask` procedure in Listing 2. That task-switching `JMP` finds the address of the target instruction in the TSS selector, which is why the 32-bit offset address in `ThisTaskPtr` is always zero.

The `DemoTask` code has no special setup, merely loading the `EDX` register and beginning an endless loop. After a

few instructions, the CPU calls `TaskDispatch` and once again saves the registers and return address on a stack, this time near 00129000.

Now the magic happens.

The task dispatcher swaps `ThisTaskPtr` and `NextTaskPtr` again, restoring `Kerne1`'s TSS selector to `ThisTaskPtr`. The task switch restores all of `Kerne1`'s registers including `SS:ESP` and `CS:EIP`. The first instruction executed in the `Kerne1` task is the one immediately following the task-switching `JMP` in `TaskDispatch`. The `Kernel` code picks up where it left off, twiddles the port bits, and executes the `RET` using the return address and registers stored on the stack it set up before the first task switch.

As far as `Kerne1` can tell, it entered `TaskDispatch` and exited normally because the CPU hid all of the task-switching hocus pocus inside the `JMP` instruction. If you prepared the TSS fields and selectors correctly, the task switches will be transparent to the users. If you screw up, you get a protection exception.

Following the CPU's EIP register during TaskDispatch reveals no discontinuity during the second task switch. Execution proceeds smoothly through the indirect JMP just as though it didn't exist, even though the CS register changes to reflect the new TSS. If you can keep that straight in your head, you'll do well at this multitasking stuff.

The process continues when

Kernel calls TaskDispatch again. The JMP instruction switches back to DemoTask, which returns from its version of TaskDispatch using its own stack. Because neither task can see the other's stack, there is no way to return from an incomplete call or restore the wrong registers. No trace of either task shows up in the other's stack, which should be enforced by placing them in their own nonoverlapping segments.

You've probably read articles describing how to pull off this stunt in real mode. No matter how elaborate the code, it always boils down to a few key lines, typically in assembler, that swap the stacks. In protected mode, all that trickery collapses into one instruction because the CPU is on your side...as long as what you want to do matches that CISC silicon, of course.

Now that we have the taskettes under control, let's look at the deceptively mundane task of displaying readable values on the serial port, VGA screen, and LCD panel.

FORMATTING THE FIELDS

The TSS dumps in Figure 2 were produced by a simple `sprintf()` clone I wrote to make formatted output easier. Although `StrFormat` doesn't include all the bells and whistles, it can display decimal and hex numbers, ASCII characters, and C-style strings. You control the output field width, value alignment, and zero or blank fill. That's enough for now.

```
TSS Sel=1000 Base=00130000 Name [FFTS Kernel 1
Backlink=0000 LDT=0000
CS:EIP=0000:00000000 EFLAGS=00000000 CR3=00000000
SS:ESP=0000:00000000 EBP=00000000 IOMap=0000 Trap=0000
DS=0000 ES=0000 FS=0000 GS=0000
EAX=00000000 EBX=00000000 ECX=00000000 EDX=00000000
EDI=00000000 ESI=00000000
SS:ESP 0/0000:00000000 1/0000:00000000 2/0000:00000000

TSS Sel=1010 Base=00130200 Name [Demo Task]
Backlink=0000 LDT=0000
CS:EIP=0030:00001794 EFLAGS=00000000 CR3=00000000
SS:ESP=0028:00006FFC EBP=00000000 IOMap=0000 Trap=0000
DS=0018 ES=0000 FS=0020 GS=0000
EAX=00000000 EBX=00000000 ECX=00000000 EDX=00000000
EDI=00000000 ESI=00000000
SS:ESP 0/0000:00000000 1/0000:00000000 2/0000:00000000
```

Figure 2—Even after they're set up for the first task switch, the two Task State Segments (TSS) contain mostly zeros. The first TSS requires less initialization because it will receive the CPU state during the first task switch. The second TSS must contain valid CS:EIP and SS:ESP fields. The selectors for these segments reside in the GDJ.

One 32-bit protected-mode gotcha will hit you in the knees when you pass segment information on the stack. In 16-bit mode, the CPU pushes constants as 16-bit values. In 32-bit mode, it extends them to a full 32-bit value.

In either mode, the CPU pushes segment registers as 16-bit quantities; it does not pad them to 32 bits to maintain DWORD stack alignment. If you are simply saving and restoring segment registers around a function call, this is no big deal. If you are passing parameters to a function, it can kill you.

Here's the catch. `StrFormat` must know the size of each value on the stack so it can step through them. The default size is four bytes, as you might expect, which will not work with `PUSH DS`. In that case, `StrFormat` will step halfway through the next parameter on the stack when it increments its internal pointer by four bytes.

The ANSI-standard-library `sprintf()` format string includes several size specifiers. I implemented the `h` specifier to identify 16-bit quantities. Therefore, the format string `"%x"` processes a 32-bit stack entry and `"%hx"` handles a 16-bit entry.

The catch comes when you use the same format string in two places. Suppose you push either a segment register (`PUSH DS`) or the corresponding constant-segment selector (`PUSH GDJ_DATA`). In the first case, the stack

gains a 16-bit segment register and, in the other, it gets a 32-bit constant that's numerically equal to the segment register. Regardless of whether you use `"%04x"` or `"%04hx"`, you lose in one case or the other.

This problem is not unique to `StrFormat`. Any function that expects a segment value must know whether the stack will hold a WORD or a DWORD quantity. What's actually there depends on how you

phrase the CALL instruction.

TASM sports a `PROCDESC` directive that specifies the size of procedure arguments and enables simple type checking for procedure calls. Regrettably, `PROCDESC` doesn't work as documented and the bugs render it useless. I guess it's another one of those neat features that might, just possibly, be cleaned up in the next release. Until then, be careful.

RELEASE NOTES

The code this month creates two TSSs and dumps their contents to the serial port using `StrFormat`. It then enters an endless loop switching between the tasks while displaying a count on the FDB's LEDs. You can view the relative times on your system by watching the three low-order bits of LPT1 on a scope.

Next month we'll introduce several more taskettes, activate their LDTs, and start some memory management. □

Ed Nisley, as Nisley Micro Engineering, makes small computers do amazing things. He's also a member of the Computer Applications Journal's engineering staff. You may reach him at ed.nisley@circellar.com or 74065.1363@compuserve.com.

IRS

- 413 Very Useful
- 414 Moderately Useful
- 415 Not Useful

Getting By With Next To Nothing

Micro-power Wake-up Control

You'd think
with this
lean,
mean,
wake-up machine that
there'd be no eggs and
bacon. But, with the
right power management,
a lot can be done on
very little.

FROM THE BENCH

Jeff Bachiochi

a

lthough many micros can withdraw into power-down or sleep modes, current savings is never quite what you wish. Dwindling to 10 mA sounds good until you do the math for extended hours of use. At 8766 hours per year, it would require an 88-Ah battery to operate the task (discounting battery shelf life). In a real data-logging situation, the logging task often takes less time than it does to reset the processor.

Why keep the whole system powered, even in a so-called low-power or sleep mode, when significant savings can be obtained by switching the system off between tasks?

There are a few issues I would like to examine here: power-supply control, quiescent current, power-on reset timing, and dead timing.

ON/OFF REGULATION

Back in INK 22, I introduced the Toko linear regulator with integrated

on and off control. Today Toko's line has expanded into a complete selection of these regulators from 2.0 V to 5.0 V, which can supply up to 100 mA of output current.

Other manufacturers are now offering various configurations of logic-switched regulators. For instance, Linear Technology, Sharp, and Seiko offer linear regulators. Maxim has a switched-capacitive, step-up regulator. Linear and Maxim both present switching-style regulators. [Note this is not a complete list by any means. It just includes the parts I've used.]

Most regulators have fairly large off-state quiescent currents, restricted input-voltage levels, or small output capability, which makes them poor choices for this circuit. Instead, I want the perfect regulator—no off-state quiescent current, a wide range of input voltage, capable of infinite current, and costing under a buck.

OK, I'm willing to give in a little.

Let's say it has to run off an RC-style NiCd pack, a camcorder battery, or a small gel cell and be capable of up to 1-A output current at 5 V. A 7805 or low-dropout 2940-5 fits these criteria, but it isn't logic controlled. If you refer back to Steve's article in INK 15, you'll see that the input voltage to the linear regulator can be switched on and off with a few additional transistors providing logic-level control and very little operational current in the off state.

Sharp's PQ05RA1 is a linear low-dropout regulator with a logic state

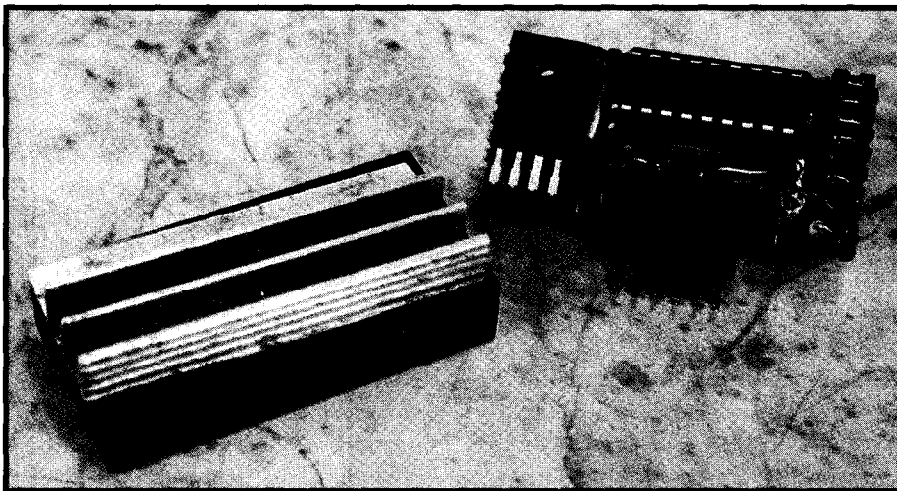


Photo 1—The micro-powered wake-up controller fits neatly inside a small case. A single connector on the bottom is used to exchange information with the outside world.

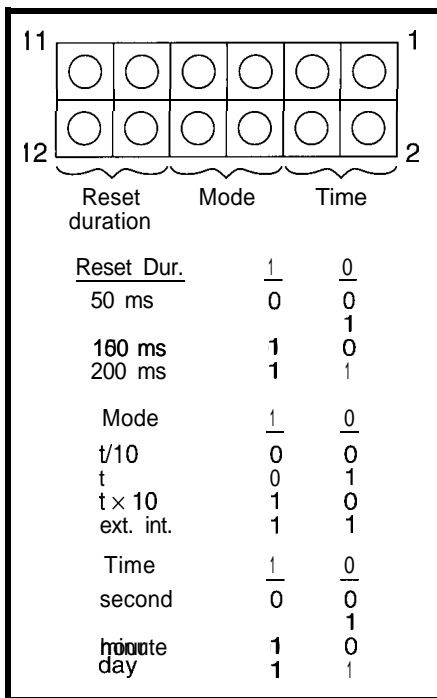


Figure 1—Three sets of jumpers inside the power controller are used to configure the unit's reset duration, timing mode, and time interval. For example, puffing jumpers between pins 1 & 2, 5 & 6, and 9 & 10 would select a 150-ms reset pulse and would tell the unit to awaken every 10 hours.

controlled on/off. Not only will this device regulate with an input as low as 5.5 V or as high as 35 V, but it can also supply a full 1 A of current, not to mention the standard over-current and thermal cutoffs. In the off state, it requires less than 1 μ A of quiescent current.

I guess manufacturers do listen to us after all.

DEAD CURRENTS

Off-state quiescent currents alone will not make this periodic kick-start regulator successful. Some circuitry must remain active to provide the wake-up signal. I wanted this circuitry to use a timebase with a wide variety of useful periodic timings. This is generally not the case with a simple oscillator or divider. You might get 1, 2, 4, 8, 16, 32, 64 seconds and beyond, but I don't consider this universally handy. Instead, I'd prefer a second, minute, hour, and day as the units or maybe even a wake-up on external interrupt.

To solve for these broad requirements, I'll explore using a small PIC processor in a slow, low-power mode.

A 32.768-kHz crystal gives the slowest stable timebase I know of and is easily divided into useful pieces. The only trouble with slowing down the PIC's clock is that the number of execution cycles per second also decreases. (I'll get into more on this latency problem later.) To reduce current consumption even more, the PIC is run at reduced voltage while the external system is dead.

This brings up another problem. We need a regulator for the PIC. Maximum voltage in LP mode is 6.0 V (min = 2.5 V). This regulator must have ultralow quiescent voltage (it's on all the time). However, it doesn't have to supply much current (except to the PIC). A zener diode would require a few milliamps of current whereas a standard regulator takes 100 mA. Enter the micro-power regulators like Seiko's S8 1233PG. Although operating current is about 3 μ A, it supplies more than 10 mA at 3.3 V.

I chose a 3.3-V regulator so I could add a Schottky diode drop to the PIC and still end up with greater than the minimal 2.5 V. A second Schottky is added from the switched regulator's 5-V output. When the 5-V regulator is switched on, the PIC is then powered by 5 V and is logic-level compatible (more on this later).

ON/OFF DEAD TIME

For the periodic dead timings, I want those nice round increments: 1 second, 1 minute, 1 hour, and 1 day. To put some meat on the bones, I added "times 10" and "divide by 10" selections. This gives 100 milliseconds, 1, 6, and 10 seconds, 1, 6, and 10 minutes, 1, 2.4, and 10 hours, and 1 and 10 days. (Although 100 ms could be too short a time period depending on the reset and task-execution times, I left it in anyway.)

When the dead time times out, the 5-V regulator is turned on. When the task completes, the task raises an output bit designated as the off control. This bit signals the PIC to remove the power and enter another dead cycle. This arrangement keeps the power to the task on for the shortest possible time, thereby providing real current savings.

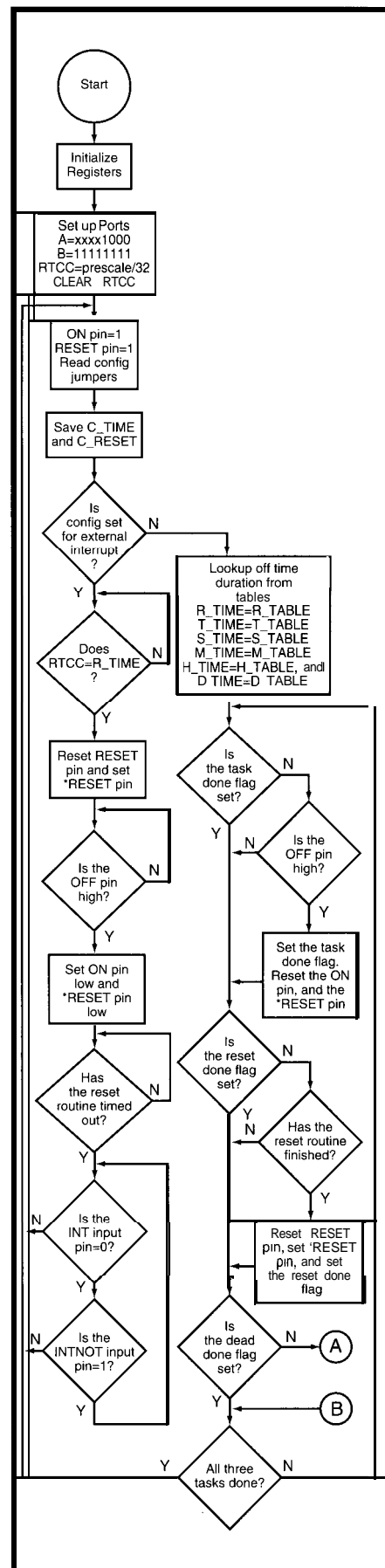


Figure 2a—The main control loop flowchart consists of power on, reset, and power off.

MICROPROCESSOR RESET

Many microprocessors use an RC time constant to delay program execution until power has been established and the oscillator is running. These timings are generally longer than necessary just to be safe when using wide-tolerance parts. The task can often be completely executed in less time than the RC's time constant. So, tighter control of the micro's reset could save valuable full-current operating time.

Two of the PIC's pins are used as RESET and complementary *RESET outputs. The reset duration is jumper selectable from 50 to 200 μ s in 50- μ s increments. This flexibility gives different reset timing depending on the microprocessor and power-supply slew rate. You need to know the minimal reset time for your external system to take full advantage of one of these PIC outputs.

USER SELECTIONS

Three sets of jumpers help make user selections simple [see Figure 1]. The first set chooses a dead-time period. The second selects the mode of the dead time. The final set selects the reset time period. An installed jumper presents a logic 0 to the chip, while a removed jumper allows the input to be pulled to a logic 1.

The only odd-ball setting occurs when the mode equals 11. This is an external interrupt mode and wakes the dead system only when INT is high or \bullet INT is low. These input pins are *not* actual interrupt pins on the PIC, but are polled in a tight loop of three 2-cycle instructions. Although under 750 μ s (with the PIC running at 32.768 kHz), it is a far cry from the 1.5 μ s possible when the processor is running at 8 MHz.

GO WITH THE FLOW

The flowchart in Figure 2 follows this simple code. Initialization begins

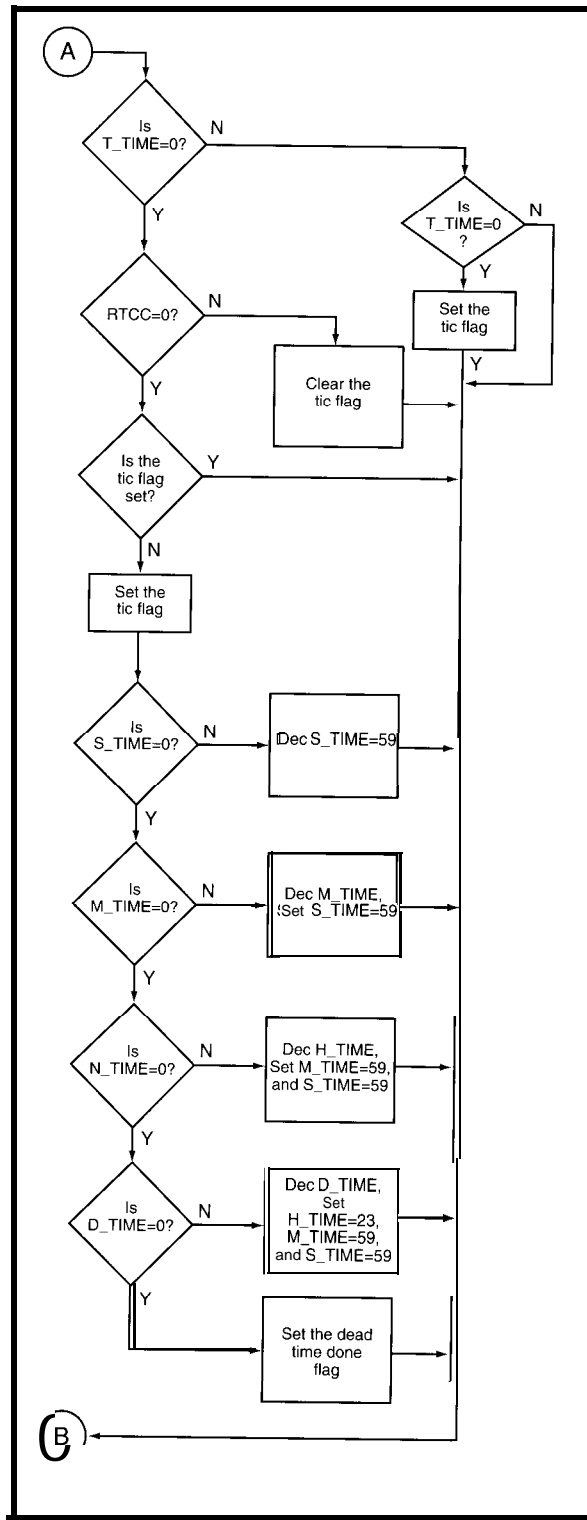


Figure 20—A side loop in the power-controller flowchart keeps tabs on the dead time.

with a clearing of all registers and the proper setup for the twelve I/O bits of port A and B. The nybble port, A, uses the first three bits (pins) as outputs (RESET, *RESET, and ON) along with a fourth as an input (OFF).

- Port B uses all bits as inputs (INT, \bullet INT, TIME0, TIME1, MODE0,

MODE1, RESET0, and RESET1). The PIC's on-chip timer, the RTCC, is set for a prescale of 32. Using a 32.768-kHz crystal gives you an instruction time of 122.0703125 μ s or $4 \times \frac{1}{32768}$. Factoring in the prescaler gives an RTCC tick of 3.90625 ms or $32 \times 122.0703125 \mu$ s. The RTCC register counts from 0 to 255 (eight bits) which is 1 second or 256×3.90625 ms. A nice round number, eh? And, if we don't change the RTCC, the overall timing should remain as accurate as the crystal without having to pay much attention to the number of instruction cycles.

The main loop starts by raising the ON and RESET pins. This turns on the regulator and holds RESET high and *RESET low. The configuration is read to determine whether the branch to take is wake-up (set time) or external input (interrupt). The reset time is read from a four-place R_T A B L E based on the configuration jumpers connected to the RESET0 and RESET1 input pins.

If the mode bits are configured for external interrupt, the RTCC is read, and this number is added to the reset time (R_T I M E). When the RTCC reaches this count, RESET is lowered and *RESET is raised. The external system is now out of reset and can proceed with its task. When done, it signals the PIC by raising the OFF line. The PIC now waits indefinitely for either a low on *INT or a high on INT.

If the mode bits are configured for a dead time, the RTCC is read and R_T I M E is adjusted as described above. In addition, the time and mode bits point to one of twelve positions in each of five tables. These tables hold the appropriate timing values for all the possible selections from $\frac{1}{10}$ second to 10 days. (The special case where time equals

Listing 1-This BASIC-52 program monitors the temperature and storage position of delicate "malarkey" during shipment and then indicates the end of the task.

```

10 A = XBY(2000H)*256 + XBY(2001H)
20 B = (255-XBY(2002H))*256 + (255-XBY(2003H))
30 IF (A<>B) THEN GOTO 170: REM Error so leave
40 IF (A>7FFEH) THEN GOTO 170: REM Out of space so leave
50 XBY(A) = XBY(0E000H): REM Sample ADC and store
60 A = A+1
70 XBY(A) = PORT1.0R.0COH: REM Sample tilt and store
80 A = A+1
90 B = INT(A/256)
100 XBY(2000H) = B
110 B = 255-B
120 XBY(2002H) = B
130 B = A-(256*INT(A/256))
140 XBY(2001H) = B
150 B = 255-B
160 XBY(2003H) = B
170 PORT1 = PORT1.0R.80H: REM Set bit 7 high for OFF signal
180 GOTO 170

```

$\frac{1}{10}$ s adjusts T_T I ME for fractions of a second just like R_T I ME.)

The timing loop starts by checking for an off signal sent by the external system when it finishes with its task. If done, the ON and RESET pins are lowered, and the task-done flag is set. Next, the reset timing is checked. If it is done, RESET is lowered, *RESET is raised, and the reset-done flag is set. Finally, the dead time is checked. Ordinarily, all registers (seconds through days) are checked once a second for 0. If all are 0, the dead-timeout flag is set. The special case of C_T I ME equalling zero indicates fractions of a second and flags the processor to check only the fractional portion. If any of the timing registers is not 0, they are all adjusted each time through the timing loop (once per second] like a digital clock counting backwards.

Note that I could have used a total number of seconds for each time period instead of a seconds, minutes, hours, and days format. Using just seconds would save one register, a bunch of code space for tables, and the complexity of decrementing multiple values (i.e., 60 s, 24 h). But, while simulating and debugging, I like to see the registers showing me the actual time remaining and not just some abstract total number of seconds. If that makes me an immoral programmer, so be it.

Once all the flags are set, the timing loop is finished. Control jumps back to the main loop.

TEST CASE

Let's say we have this case of malarkey which needs to remain under

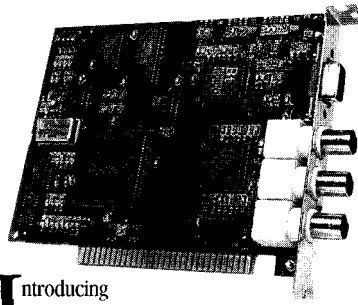
40°F. This volatile stuff spoils if it is not stored upright. The task processor shipped along with the malarkey samples the ambient temperature and tilt of the case to ensure the requirements are met by the delivery company. The journey takes two months. Logs, taken once every 10 minutes, account for over 14,000 samples.

Each sample consists of two bytes-one for temperature and one for tilt. The temperature is measured using a silicon temperature sensor. The sensor's output is 10 mV per Fahrenheit degree. An 8-bit, O-1-V A/D converter registers 0-100°F in less than 0.5-F" increments (plenty of accuracy).

The tilt sensors take six bits-one for each side the case can be left on. By setting the upper two bits of the tilt sample high, the two sampled bytes can be discerned because the upper bit of the temperature should never be high (unless the temperature exceeds 50°F).

Listing 1 offers a short program written in BASIC-52 which checks NVRAM for a legal address (stored

PRECISION FRAME GRABBER FOR ONLY \$495*



Introducing the CX100 precision video frame grabber for OEM, industrial and scientific applications. With sampling jitter of only ± 3 ns and video noise less than one ISB, ImageNation breaks new ground in imaging price/performance. The CX100 is a rugged, low power, ISA board featuring rock solid, crystal controlled timing and all digital video synchronization. A Software developers will appreciate the simple software interface, extensive C library and clear documentation. The CX100 is a software compatible, drop-in replacement for our very popular Cortex I frame grabber. A Call today for complete specifications and volume pricing.

ImageNation Corporation
Vision Requires Imagination
800-366-9131

— CX100 FEATURES —

- . Crystal Controlled Image Accuracy
- . Memory Mapped, Dual-Ported Video RAM
- . Programmable Offset and Gain
- . Input, Output and Overlay LUTs
- Resolution of 5 12x486 or Four Images of 256x243 (CCIR 512x512 & 256x256)
- . Monochrome, 8 Bit, Real Time Frame Grabs
- . Graphics Overlay on Live or Still Images**
- . External Trigger Input
- . RGB or B&W, 30 Hz Interlaced Display
- . NTSC/PAL Auto Detect, Auto Switch
- . VCR and Resettable Camera Compatible
- . Power Down Capability
- . BNC or RCA Connectors
- Built-In Software Protection**
- 63 Function C Library with Source Code
- . Text & Graphic Library with Source Code
- . Windows DLL, Examples and Utilities
- . Software also available free on our BBS
- . Image File Formats: GIF, TIFF, BMP, PIC, PCX, TGA and WPG

** THESE OPTIONS AVAILABLE AT EXTRA COST

* \$495 IS DOMESTIC, OEM SINGLE UNIT PRICE.

P.O. BOX 276 BEAVERTON, OR 97075 USA PHONE (503)641-7408 FAX (503) 643-2458 BBS(503) 626-7763

both as address and complement for security's sake) before taking any samples. After storing the samples, the address pointer (and its complement) is updated for the next sample. The OFF pin is raised signaling the PIC to drop power.

The RS reset time of my RTC52 is 55 ms. The task execution is 88 ms. I save little by using the PIC's RESET output pin (however, the minimal reset time necessary for the RTC52 is about 20 ms). Current consumption during this period is 101 mA, which is about 39 μ Ah per sample (task). At 14,400 samples, that's 0.56 Ah of battery current needed for the external system.

The dead-timer circuitry shown in Figure 3 requires 25 μ A of current. Since it will be running 24 hours per day for the 60 days, it requires 36 mAh of battery current.

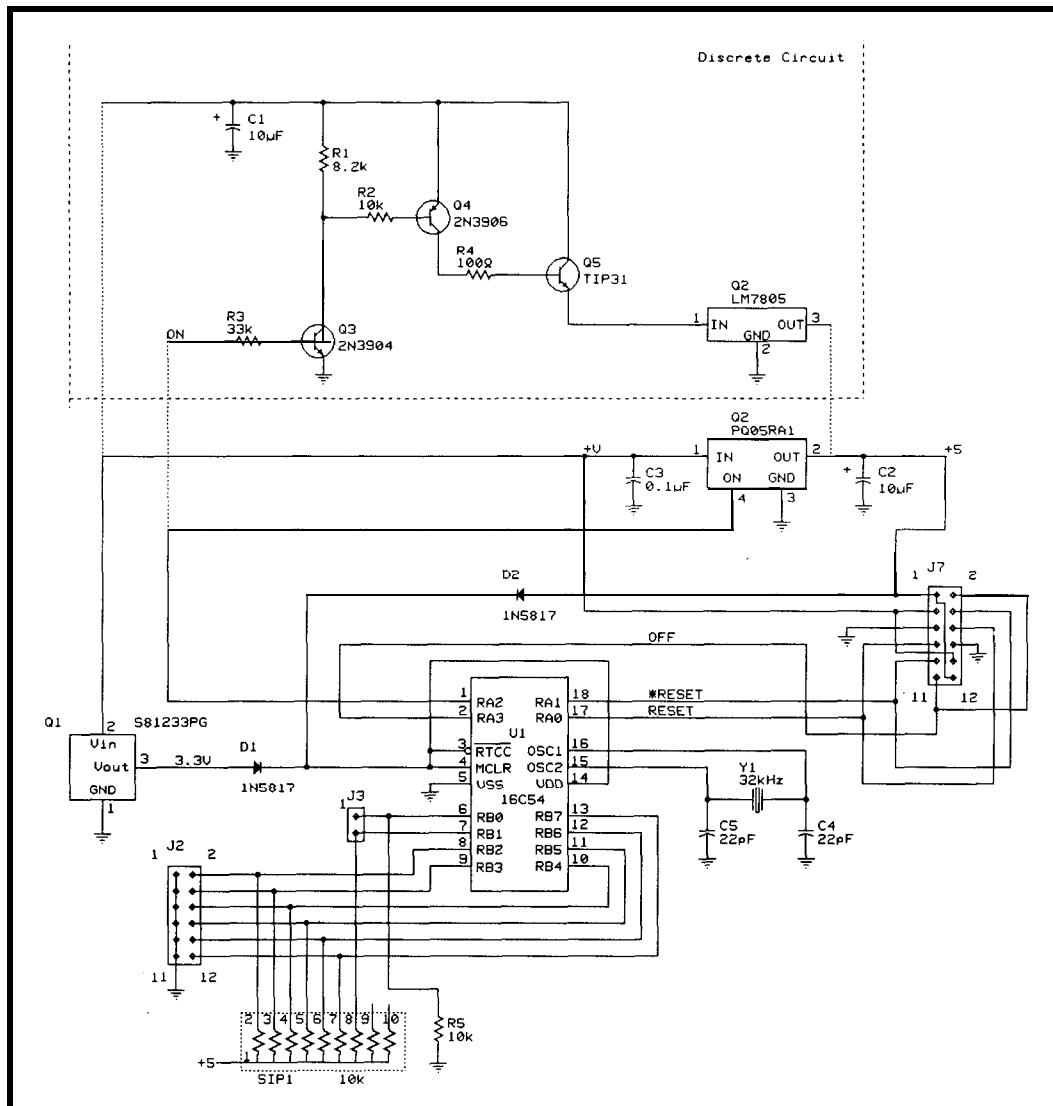


Figure 3—The micro-powered wake-up controller is based on a PIC running at 32.768 kHz. A PQ05RA1 low-dropout regulator helps keep power use to a minimum.

CONCLUSION

If you work out all the calculations, 720 Ah is required to run the external system for 60 days. In sleep mode, the external processor alone would require over 14 Ah. Either power usage is unacceptable for battery-powered systems.

But, with a little periodic stimulus, this power consumption can be reduced to a comfortable level of less than 0.6 Ah—all this from a system which normally requires 0.1 A of current for each hour it's used. \square

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

CONTACTS

Linear Technology
1630 McCarthy Blvd.
Milpitas, CA 95035
(408) 432-1900
Fax: (408) 434-0507

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

Microchip Technology, Inc.
2355 W. Chandler Blvd.
Chandler, AZ 85224-6199
(602) 786-720
Fax: (602) 899-9210

Seiko Instruments, Inc.
Semiconductor Products Group

1150 Ringwood Ct.
San Jose, CA 95131
(408) 433-3208
Fax: (408) 433-3214

Sharp Electronics Corp.
Microelectronics Group
5700 NW Pacific Rim Blvd., Ste. 20
Camas, WA 98607
(206) 834-2500
Fax: (206) 834-8903

Toko America, Inc.
1250 Feehanville Dr.
Mount Prospect, IL 60056
(708) 297-0070
Fax: (708) 699-7864

IRS

416 Very Useful
417 Moderately Useful
418 Not Useful

HOME AUTOMATION

Building & Control

Premiere

*Use Your Home's Existing Cable to Network
Your A/V Equipment and Your PC*

Add Hard-wired Devices to the HCS II

Design the "Ultimate" Hand-held Remote

Make Your PC into a Telephone Receptionist



A Quarterly Bonus Section of **CIRCUIT CELLAR INK**

Contents

Page

60

Multimedia Home Networks

by David Gaddis

69

HCS Hard-wire Control: Back to Basics

by Steve Ciarcia

81

A Different Set of House Keys

Making the Most of a Small Keyboard

by Jeff Fisher

87

Computer, Get That Phone

A PC-based Voice-Telephone Interface

by Robert M. Luzensky & Jack Ivey





F

or many years, custom audio and video installers have created whole-house entertainment systems for the rich and famous. During the last few decades, the cost of these systems fell within reach of an average

millionaire, but it still remained above the resources of most middle-class families.

As multimedia emerges into the home, many people have learned that they can now watch television shows or listen to a CD from their computer. But, this is only scratching the surface. How would you like to create a first-class, whole-house entertainment system based on your multimedia computer and on a budget you can afford?

With a few small additions, you can have VCRs or laser-disc players in other rooms display on your multimedia computer. You can also see, hear, and control the CD-ROM in your multimedia computer from any television in your home. You can even create a video-intercom system that lets you see and hear users from other rooms in your house through a multimedia computer or television. You can install these features in an afternoon using existing cable.

To understand your options, let's first take a look at what's going on today. Those who install or use entertainment systems in homes are involved with one of three possibilities: distributed, centralized, or networked entertainment. Understanding these three possibilities can help you with future decisions.

DISTRIBUTED ENTERTAINMENT

Once upon a time, the home's entertainment system was limited to a single antenna or cable connection to the one radio or television located in the living room. Then, someone bought televisions for other rooms, connected them to the incoming source through splitters and cable, and created what we call a distributed entertainment system (see Figure 1).

In this concept, coax cable is used to distribute the incoming entertainment signals to output devices (televisions) in any of all rooms. With the aid of a video adapter, multimedia computers can be added to the list of output devices, so you can watch Star Trek on your computer. While it represents a step up from an old-fashioned, single-outlet system, it still falls short of many of the features and advantages that are available.

Multimedia Home Networks

CENTRALIZED ENTERTAINMENT

For only a few \$10 thousand, many custom audio and video dealers will install a centralized entertainment system in your home. In this environment (see Figure 2), one room in the home is selected as the entertainment center where a variety of entertainment products are installed.

Using distribution amplifiers, miles of special-purpose wiring, and remote speakers, this system broadcasts the source in the entertainment center through the home. In other words, a CD playing in the entertainment room can be listened to while in any other room in the home. You could compare this system to an older mainframe computer where you would pay a ton of money to have all the processing done in one location.

Compared to a simple distributed system, a centralized entertainment system provides a lot of features. With the addition of wall-mounted volume controls, infrared repeaters, and more wires, the user can even control the entertainment equipment from remote rooms.

However, distribution is still limited to the equipment in the entertainment center. Remote users have to access the CD in the player of the entertainment room. Furthermore, rooms can end up with several sets of speakers. One set of speakers is dedicated to the house-distribution system while another set is used by the multimedia computer or other audio and video equipment in that room.

NETWORKED ENTERTAINMENT

The third system is often referred to as *networked entertainment* or *home networking* (see Figure 3). In this environment, the output from your computer and entertainment devices can be distributed throughout the home. The user can access any of the devices from any room.

In many homes, the wiring is already in place for a networked entertainment system. You can use the

DAVID GADDIS

Forget rewiring. Use your existing coax cable system to create a first-class, **whole-house** entertainment system that includes a multimedia computer and is on a budget you can afford.



same coaxial cable that distributes the entertainment signals from the outside world. In simple terms, you connect the output of your computer and entertainment devices to the cable and then tune the receivers in other rooms to the channel that displays the device you want to access.

In addition to providing greater control and flexibility, this concept is easier to install and costs less than a centralized entertainment system. To set it up, you can start with your present distributed system and add a few new components.

Let's take a closer look at this system.

COMPUTERIZED TELEVISION

Multimedia computers can be added to a distributed entertainment system by adding a video-adaptor card to your computer and attaching the cable system to the F connector on the video-adaptor card. This setup lets you watch any television show on your cable from the computer.

The software that comes with the video adapter also offers certain controls. You can resize the picture and move it into a corner (you don't have to miss your favorite show while working in other applications), or use a capture command to freeze a frame. Captured frames can be imported into other software where they can be manipulated and used.

BROADCASTING SIGNALS

The video-adaptor manufacturers are proud of the fact that VCRs and video cameras can be attached to their board as an input device. However, if you follow their general instructions, you usually have to relocate your VCR or video camera to a location near the computer. They never point out that you can connect the VCR to the cable so you can watch VCR-played movies on your computer or any other television in the home.

To do this, however, there are limitations to overcome. The outputs of most entertainment devices such as your VCR are usually designed by the manufacturer to be limited to either television channels 3 and 4 or line-level audio and video (NTSC). Ob-

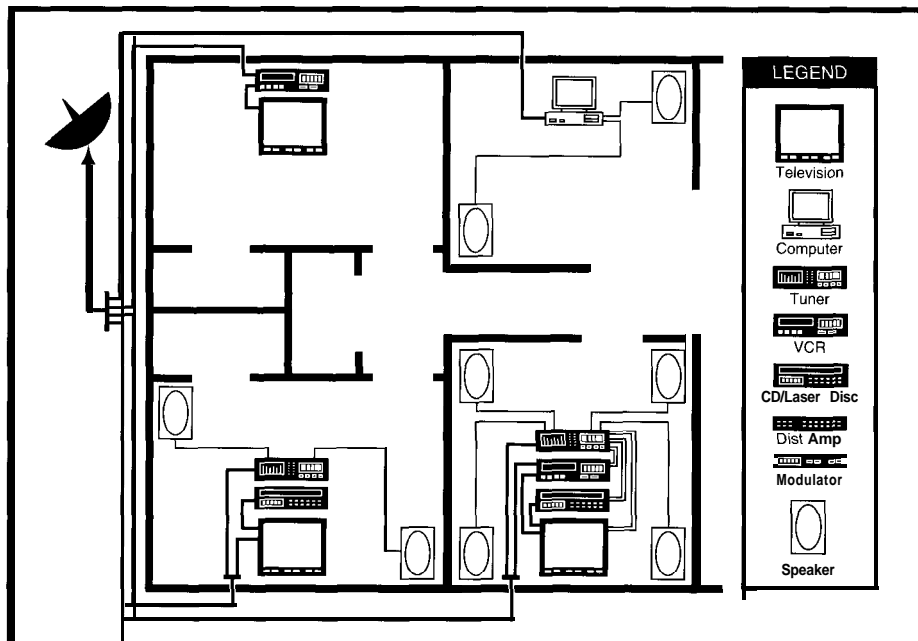


Figure 1: In a distributed entertainment system, incoming entertainment signals are split and distributed through coax cable. The computers and entertainment equipment in each room are used as output devices only.

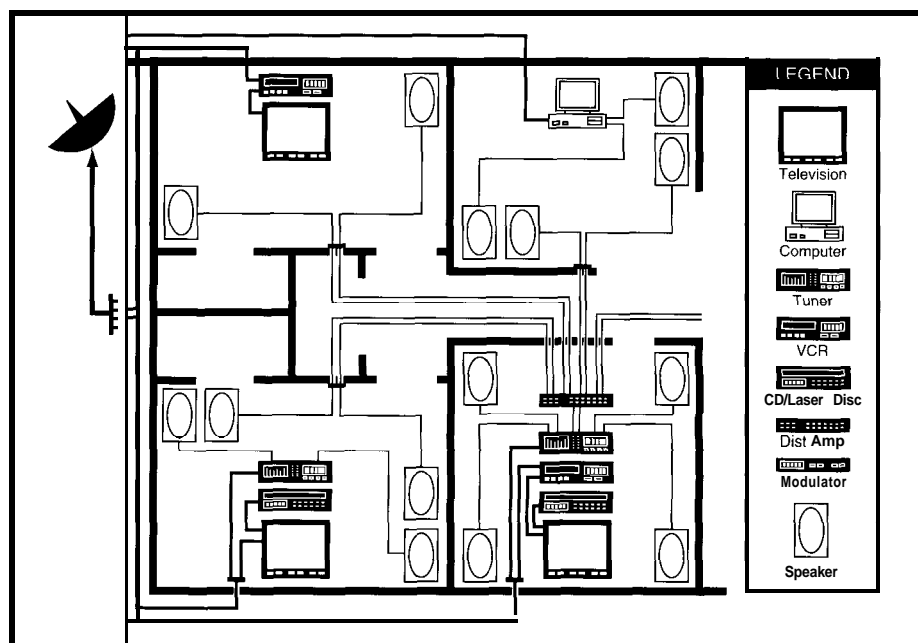


Figure 2: In a centralized entertainment system, the entertainment equipment in one room is connected to a distribution amplifier and distributed to speaker outputs in each room. This requires extensive wiring and additional speakers.

viously, if multiple devices are broadcasting simultaneously, they cannot use the same frequency without the signals interfering with each other.

For example, if you connected the output of your VCR directly to the cable, it would interfere with channels 3 or 4 already there. The televisions and computers would receive a mess.

Perhaps someday, manufacturers will offer a wider selection of



outputs for entertainment products. For now, this condition can be addressed by the use of a device known as a *modulator*: A modulator, pictured in Photo 1, is a small electronic device that receives a signal most entertainment devices output. The modulator changes the signal's frequency so that the device's output can be broadcast in another channel.

By connecting a modulator to the audio and video outputs of your VCR (or any

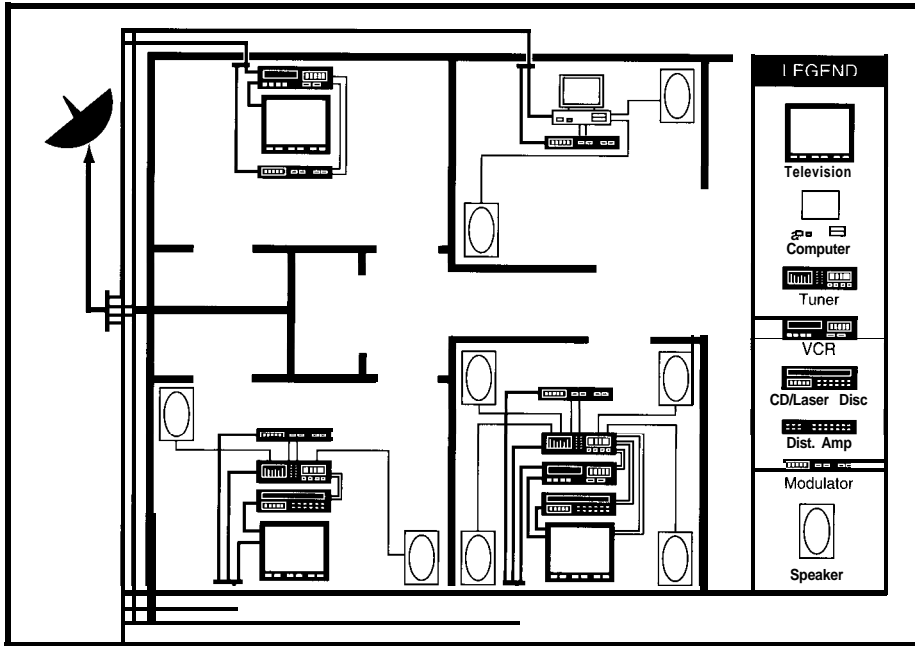


Figure 3: In a network entertainment system, the outputs of entertainment devices in each room are connected to modulators and placed on a distributed entertainment system. You can tune to any device from any room without special wiring and speakers.

other entertainment device), the signal coming out of the modulator can be tuned to an unused channel between 2 and 120. You can then connect the modulator's output to the cable system without interference.

Suppose you are connected to a cable system which provides programming on the

first 50 television channels, and you own a VCR and a laser-disc player located in different rooms. Modulators can be connected to each of the devices, tuned for output as channels 52 and 54, then reconnected to the cable system. Now, from your computer or any televi-



sion in the home, you can access all the regular cable channels and receive a program from the VCR on channel 52 or watch a laser-disc movie by changing to channel 54.

SELECTING A MODULATOR

Modulators are available with several options and styles. They are available for connection to one, two, or three input devices and range in price from \$40 to \$650. The lowest-cost modulators are designed for mono or a single-audio channel. They are ideal for such things as video cameras, which only use one audio channel.

However, for distributing your VCR or laser-disc movies, you may want to opt for a stereo modulator, which includes both left and right audio channels. Many also include a digital readout displaying the output channels on the front panel. Lower cost models do not include this option. One of the most significant considerations involved with the selection of a modulator is based on the needs of the system.

Determining the type of modulator you need requires a somewhat historical perspective. A long time ago, the FCC allotted frequencies for different television channels. At that time, 55.25-211.25 MHz was considered very high frequency (VHF) and all of the networks were covered by channels 2-13. They then allotted the bandwidth of 211.25-471.25 MHz for other services such as ham radio and air-traffic control. When the FCC needed more television channels, they added channels 14-69 in the ultra high frequency (UHF) range of 471.25-801.25 MHz.

When cable companies decided they wanted more than 69 channels, they squeezed channels closer together in the bandwidth typically reserved for ham radios or air-traffic control. As a result, antennas receive the channels from 14 and higher on a



ustecTM

"The wall socket of the future." *Popular Mechanics*, September '94

"The system is the first pre-wiring system that prepares the home for the future CEBus products." *Home Theater*, April '94

"The Tee-System - a wiring backbone developed to accommodate the growing digital communications needs of the home." *BYTE*, Nov. '94

"Wiring up for telecommuting -tomorrow's technology." *Interiors*, Nov. '94

"The US Tee-System - wiring at its best." *Electronic House*, Nov. '93

List of Top 50 most popular products featured in 1993. *Builder*, Dec. '93

The experts agree, homes should be Tee-wired *now*,
 Call today for more information.
1-800-836-2312 Canandaigua, NY 14424

470 South Pearl Street

different set of frequencies than the same channels are carried on cable systems (see Figure 4). This is why cable-ready televisions have a cable/antenna switch on the back. In actuality, the cable-ready switch adjusts the tuner to the different frequencies of the two systems.

It is important to understand this background since modulators are typically designed with a limited range of output frequencies. For example, some modulators have an output range of 300-468 MHz, which

INSTALLING MODULATORS

Some modulators have a front panel that extends from each side and has screw holes for attachment to the vertical rails inside a cabinet. Since you may not have a cabinet designed for this, the manufacturers also offer a countertop model which sits on top of your VCR or other entertainment devices. Jacks located on the back of the modulators are used to connect the modulator to your equipment (connecting cables are normally sold separately).

The power supply needs to plug into a nearby power outlet. Standard A/V cables

use these to prevent your neighbor from viewing your shows.

THE BEST WIRING SOLUTION

If the budget allows or you're doing new home construction, the homeowner should consider installing a dual-coax system. The key to a dual-coax system is the provision of two coax wires and connectors at each wall plate. In a networked environment, one of these coax connectors is used for output and provides the signal to televisions, computers, and other devices. The other coax connector can be used for input connections from the modulators connected to your VCRs, laser-disc players, computers, video cameras, and other entertainment devices.

Photo 2 illustrates sample components of a dual-coax system. The wall plates include a standard duplex receptacle with two coax connections and an RJ-45 jack. The eight-pin RJ-45 jack allows the user to connect a standard telephone or data device. Typically, the center two pairs of pins support two telephone lines while pins 1 and 2 are dedicated to an infrared-repeater system and pins 7 and 8 are for data.

This system uses a headend, which provides the necessary connecting points for the coax and telephone wires and is preengineered to amplify signals generated in-house before they are passed to other outlets. This amplification feature provides maximum signal quality.

The illustration also shows the communications cable, which includes a black and a white RG-6 coax cable and four pairs of low-voltage telephone wires rated Level 5. (Level 5 is the highest rating for low-voltage wires and is best for high-speed data transmission.) All of these wires are bundled together to provide convenient installations.

"WATCHING" COMPUTER

With the modulators added to your VCR and laser-disc player, you can now watch movies from your computer (or any television in the home) by simply selecting their channel. But with a multimedia computer, you access a world of CDs which don't play on your VCR. You can put a CD in your multimedia computer and watch or listen to it from any television in the home.

To see your computer on your television, you first need a video output from your computer. You accomplish this by adding a VGA-to-NTSC video-adaptor board to your computer. This board takes the entire pic-

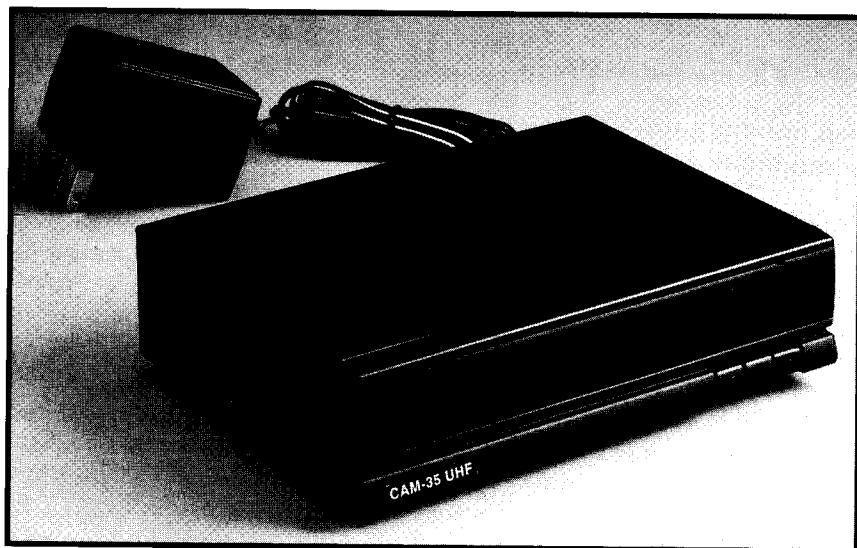


Photo 1: A modulator is used to put baseband audio and video onto an RF channel that can be tuned by any television or VCR in the house.

produces channels 37-64 on televisions set up for cable. However, the channels it produces won't be received by televisions tuned to the frequencies of antenna channels. Many people prefer to use the modulators that output in the range of 470-806 MHz, which provides channels 14-69 in an antenna-based system or channels 64-120 in a cable system.

FM modulators are also available for audio equipment. Using an FM modulator, the audio output of devices can be tuned to an unused FM radio frequency and distributed throughout a home on the cable. For instance, a CD player with an FM modulator could be tuned to an unused frequency of 101.5 FM and connected back to the cable. From any room, a user could tune the local FM receiver to 101.5 to listen to music from the CD player in the other room.

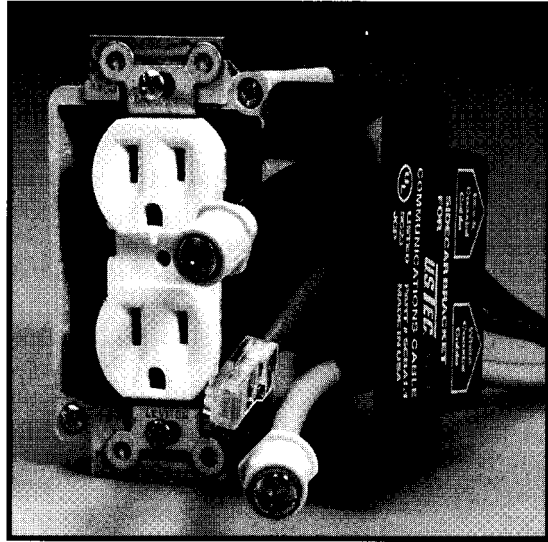
with RCA jacks connect the audio and video outputs of your equipment to the inputs of the modulator. You also need a short piece of coax cable with F connectors to attach the output of the modulator to the cable system. You may need a coax splitter to connect to a cable used by other equipment.

Modulators that include a digital readout have buttons on the front of them to select an output channel. Modulators without digital readouts commonly use a thumb wheel for manually adjusting for the best picture.

Since your neighbors may be connected to the same cable that you are using for your network, you may find that they can also watch your movies. If this is a problem, you can install a device known as a *trap* in the cable where it enters the home. Traps catch certain frequencies and prevent them from passing through. Cable companies sometimes use them to keep you from getting pay channels. You can



Photo 2a: In the US Tec wiring system, a standard single-gang box contains AC power while a sidecar bracket supports a pair of coax cables and telephone/data connections.



ture that is on your computer screen anti converts it to the analog output of your television screen. The output from this board goes to a standard RCA video jack, which can be connected to the input of— you guessed it—a modulator.

If there is an audio output on your video adapter, you can connect it to the modulator also. Although most video adapters do not include audio-output jacks, you can use the audio output of the sound card that is available on most multimedia computers.

After you tune thr modulator to your desired output channel, you can watch and listen to that computer from any television (or other multimedia computers) in the home by selecting the computer's channel.

REMOTE COMPUTER CONTROL

Now that you can see and hear the CD-ROM from any television in the home, the children can browse the encyclopedia through their televisions. Or can they? How do they select subjects from the menus on their televisions?

Many manufacturers are working on that one. Future televisions anti computers include built-in hardware that enable them to communicate with each other through the cable or even through the power-line wires they are plugged into. Among these developments, Intel anti Microsoft are working with other manufacturers to devise a system that will be introduced as a home network some time in the future.

Today, you can use another board that is available and known as an *infrared receiver board*. These boards are designed to receive infrared signals from a small, hand-held remote control that includes a keyboard and mouse buttons. This *hand-held* enables you to control your computer remotely.

However, infrared technology has limitations. Infrared remotes must be pointed at the receiver, must have an unobstructed path to the receiver, are limited to a range of approximately 35', anti do not function well in direct sunlight conditions (i.e., solariums or outdoors). Since the children are in other rooms from the computer, they

need to use an infrared repeater to carry the signals from the remote to the computer.

INFRARED REPEATERS

Infrared repeaters function just like their name implies. On one end, they receive an infrared signal. At the other end, they reproduce a similar infrared signal. So, the infrared receiver in the children's room is connected to an infrared emitter in the computer's room. When the children use the remote in their rooms, the repeater system carries that output into the other room to control the computer.

Infrared repeaters also enhance the rest of your network and extend remote controls to other rooms in your home. For example, they can carry a signal that pauses the VCR in another room or causes the laser-disc player to skip to the next track. If the children have a Nintendo game, you can connect a modulator to watch the game from other televisions (or computers). With a wireless, infrared control pad and infrared repeaters, the kids can play Nintendo against each other from separate rooms or any room that has a television or computer.

Infrared repeaters commonly use any of three methods to transmit signals into other rooms. Some of the simplest repeaters use RF to transmit signals. The infrared receiver in one room includes a radio transmitter that sends signals up to 150' and through walls. A radio receiver on the other end picks up the radio signals then reproduces the infrared signal that can be aimed at the infrared device. Such repeaters are

available as attachments to an infrared remote or as table-top devices.

Professional audio and video installers often use low-voltage wires, which are similar to telephone wires and provide a great resource for the use of infrared repeaters. The wires and RJ-45 jack described with the dual-coax system are ideal for this. When installed, you can plug infrared receivers into jacks in any room and plug infrared emitters into jacks in the rooms with devices you want to control. Alternatively, you can support the use of an infrared repeater system with an unused pair of telephone wires, or you can custom install low-voltage wiring to support this repeater.

Another method to distribute infrared commands between rooms uses the same coaxial cable carrying the television signals. This method requires special splitter/combiner devices to add and extract the infrared signals to anti from the cable in each room. If you choose this method, most amplifiers and headends are not yet designed to pass infrared signals. You need to install tiny jumper cables around them.

ADDING VIDEO CAMERAS

When the doorbell rings, many people choose to go to the door to see who is there. Some security companies like to install expensive video cameras and connect them to expen-

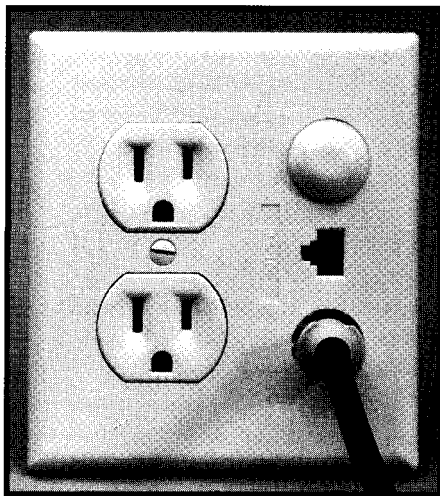


Photo 2b: With its cover plate installed, the US Tec system presents an attractive, finished look to the homeowner and keeps all wire connections in one place.

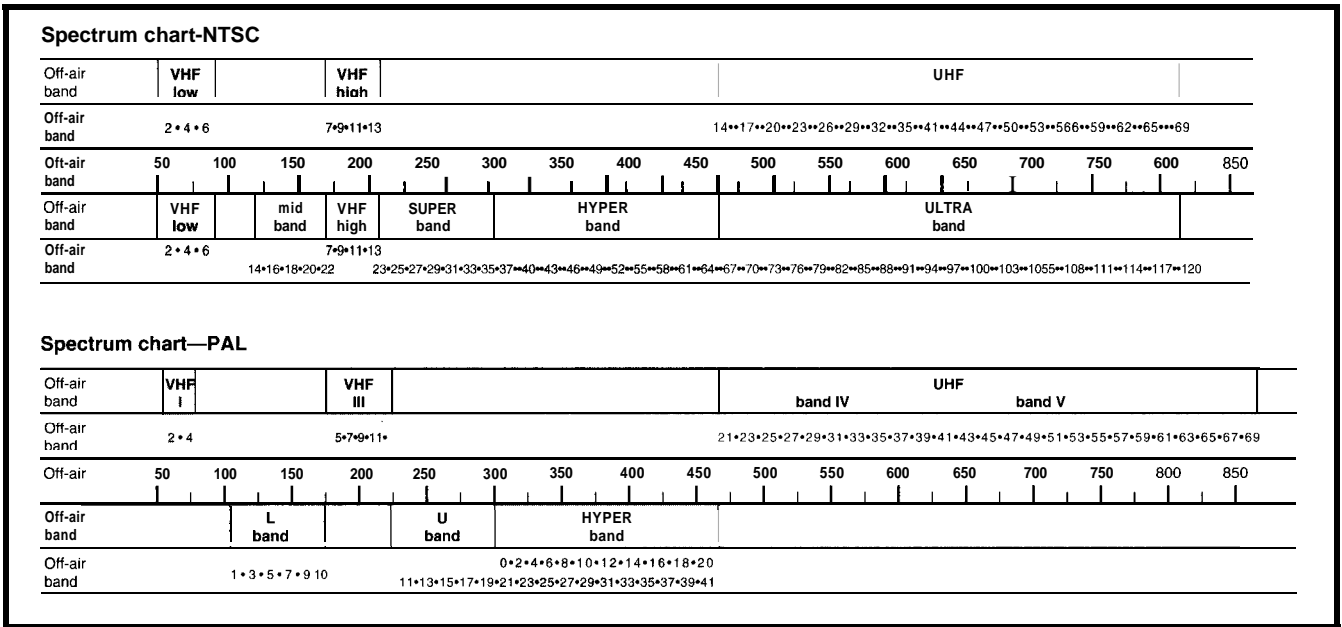


Figure 4: The allocation of television channels varies depending on whether the signals are broadcast over the air or by cable. PAL allocations (typically used in European and other countries) are also different from NTSC (used in the U.S.).

sive, special-purpose video monitors. You can do it that way, but here's another solution. Install an inexpensive video camera at the door that lets you see the visitor on your televisions.

Video cameras (often known as CCTV) are now as small as pocket pagers. They are available for both indoor and outdoor installations, and the cost has fallen to a few hundred dollars. They are designed for permanent mounting on a wall, or they can be set on top of a television, shelf, or computer. Some manufacturers also build cameras into multimedia computers and household items such as lamps, clocks, and mirrors.

You can mount one of these cameras at the front door and use a modulator to control the output so that it can be viewed as a television channel. Now when the doorbell rings, you simply change to the front-door channel and you can see and hear the visitor. Of course, the visitor **cannot** see or hear you in this situation. However, there are other devices available that enable you to pick up a nearby telephone or activate the telephone software on your computer. By these methods, you could greet your visitor through a speaker attached to the doorbell. Similar cameras placed near the backyard, pools, spas, or garages enable you to view those areas as well.

INSTALLING FIXED VIDEO CAMERAS

Temporary or portable video cameras are often used in rooms that are subject to

redecorating. On the other hand, your front door is likely to remain in the same location as are the backyard and swimming pool. For these locations, you can install a camera in a fixed and permanent position.

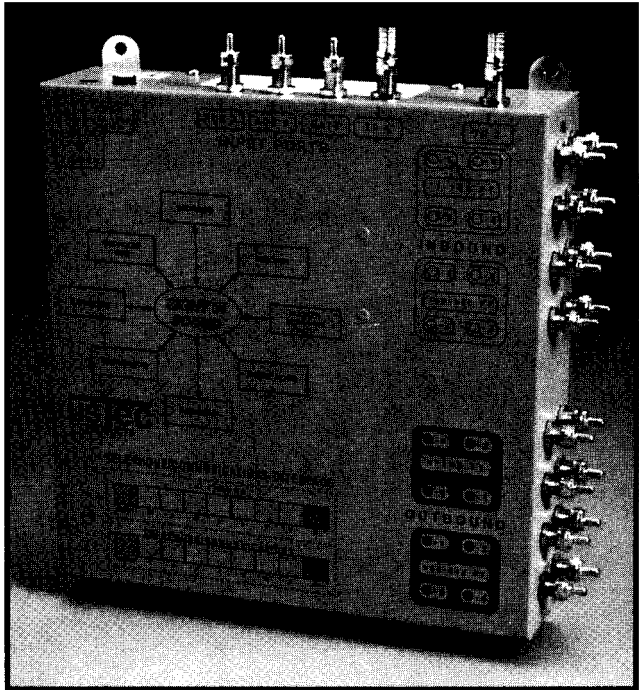
When video cameras are installed in fixed locations, there are several options for the wiring. If it is convenient, you could place the modulator nearby and connect it to the cable. However, many people prefer to install the modulators for fixed cameras in a service center or closet with other distribution equipment such as the coax headend and amplifiers. The modulator is then connected to the cable where it is installed inside the service center. The NTSC audio and video wires are run from there to the camera.

Most video cameras need a pair of low-voltage

wires to bring in DC voltage from the power supply, and they use standard NTSC audio/video cable (often with RCA jacks) to conduct the sound and picture. Video cables are preferred because they are better shielded against interference. In some cases, level 4 or 5 low-voltage wires are used for the audio and video transmission.

But, over long runs or near high-power wires or other power equipment, they can pick up interference. You can run the video wires

Photo 2c: The headend for a dual-coax system has connections for input and output coax cables and telephone wiring.



parallel to the low-voltage wires powering the camera, but avoid running the video cables parallel to power wiring. If you must cross over power wires, try to do so at right angles.

CREATING A VIDEO INTERCOM

Using the same setup as you do to see people at your front door, you can see and hear a person in specific rooms. If you set up several rooms, you can create your own video intercom system that works through your computers and televisions.

For example, say Dad is watching t&vision in the living room and Junior is working at the computer in the library. When Junior needs help with his homework, he can switch on the living-room channel and see and hear Dad on his computer. Using his remote control and the infrared-repeater system, Junior then changes Dad's television to the library channel (using the picture-in-picture if available). Now they can see and hear each other. And, if Dad wants to look at Junior's homework, Dad just

changes his television to the computer channel.

If the computer has a built-in camera, you may only need to add a modulator to get that room into your video intercom system. In the living room, family room, kitchen, and bedrooms, you can use a CCTV camera and a modulator. As you might imagine, there may be times when you don't want your children to watch what you are doing in the bedroom from their televisions or computers. Of course, you could throw a bathrobe over the camera, but you may want to include a toggle switch that turns the camera or modulator off.

Many people prefer to install fixed cameras inside the rooms because it makes a cleaner installation when the wiring can be hidden inside the walls. However, since you may want to change the camera location when you rearrange furniture, many indoor cameras are installed as temporary fixtures. As a temporary fixture, you can place the camera on top of a television or computer or you can stick it to the surface of a wall with screws, adhesive, or Velcro. In temporary installations, the modulator is often placed on a



shelf or behind other equipment, and then connected to the nearest cable outlet. As you might guess, you can get greater flexibility from this concept if you have several cable outlets in the room. Multiple room outlets are recommended in new construction or remodeling projects.

A COST ANALYSIS

Assuming that you already have all of the computers and televisions that you need, what would you expect to pay for a complete home network such as this? The final analysis is going to differ from home to home depending on many variables.

Rut, let's look at an average estimate.

You can use your existing cable. If you opt for the dual-coax system with headend, the parts cost about \$1,200 for an 8-outlet system in a 2,500-square-foot home. Because larger homes require more wire, add 25¢ for each additional square foot of your home size, and add \$10 for each additional outlet. If you want it installed by a contractor, double the cost of the materials.

The video card that allows your computer to receive television channels adds \$350 for a high-resolution model. The VGA-

**C:>everything.
you.need.to.know.
@home.automation.assn**

Join the one association dedicated to serving the fast-growing home automation industry — the Home Automation Association (HAA).

HAA is expanding the market for home automation products and services with its new *Consumer Understanding Program* ©. **HAA** includes all protocol developers, manufacturers of PC-based systems, and other major industry players. **HAA** members get the latest bottom-line news. Make sure you're in the loop. Join today.

contact.HAA.today
Internet:
75250.1275@compuserve.com
Voice: 202/223-9669

JM

ORDERS
800-477-4181
1202 Montclair Drive
Pasadena, MD 21122

JaMar Distributing

All of the products
All of the tech support
All the best prices
All at one place!

"SEND A FRIEND"
HELP 410-437-418
FAX 410-437-3757

X-10 SALE THIS MONTH ONLY

LAMP, WALL SWITCH & APPLIANCE MODULES \$119/DOZ	AUTOMATIC DRAPERY OPENER
RC6500 KEYCHAIN REMOTE 6 BASE \$19	COMPLETE SYSTEM KIT \$349
PR511 FLOOD LIGHT MOTION DETECTOR \$38	Makita
SD533 SUNDOWNER \$12	Skyline Control
PA5599 PERSONAL ASSISTANCE CONSOL \$95	INCREDIBLE SOFTWARE UPGRADE FOI
SC531 ALARM BASE (WHILE THEY LAST) \$45	X-10 CP290P \$49.95
	EASIER TO UPGRADE & SUNRISE/SUN

PANASONIC HYBRID TELEPHONE SYSTEM

	<p>KX-T7030 LCD SPEAKER PHONE \$179 KX-T7020 SPEAKER PHONE \$139 KX-T30810 \$529 KX-T30865 \$27</p>
--	---

KX-T30810 3 CO LINES, 6 EXTENSIONS, 3 INTERCOM PATHS, FLEXIBLE KEY ASSIGNMENTS
KX-T7030 LCD DISPLAY, SPEAKER PHONE, MUTE KEY, "OLD KEY, REDIAL KEY, CONF. KEY...
KX-T7020 SPEAKER PHONE, MUTE KEY, HOLD KEY, REDIAL KEY, CONF. KEY, FWD/DND KEY...
PHILIPS INTERACTIVE CD W/ COMPTON'S ENCYCLOPEDIA \$159
"EVENT CONTROL SYSTEM" DOIT ALL SOFTWARE \$250
ENERLOGIC 14000 W/LATEST SOFTWARE \$345
WHY BUY A COPY WHEN THE ORIGINAL IS THIS AFFORDABLE?
SEND A FRIEND AND RECEIVE 5% OF THEIR FIRST ORDER IN CREDIT
DEALERS WRITE OR FAX ON COMPANY LETTERHEAD

NTSC card and modulator that converts your computer output to a television channel may add \$400, and the infrared board and remote control for that computer adds another \$500. So, computer upgrades can total \$1,250.

You should include a good quality stereo modulator with each VCR, CD, and laser-disc player. Assuming a typical home has five stereo modulators at a cost of \$500 each, this adds another \$2,500 to the total.

To see the front door, swimming pool, and garage on your televisions, you can use three cameras and a 3-outlet mono modulator (usually installed in a service center). This option adds about \$1,100 for black-and-white cameras or \$1,850 for color.

For the video intercom, each set includes a camera, mono modulator, and toggle switch costs about \$500 for black and white or \$800 for color. Assuming that you want one in each of four bedrooms, two living areas, a library and kitchen, it will take eight sets or \$4,000 for black and white and \$6,400 for color.

Special thanks to USTec and Multiplex Technology for providing the photographs.

David Gaddis is president of Home Systems Network in Edmond, Oklahoma. He authored Understanding & Installing Home Systems and How To Automate Your Home. He is presently developing a television series called Intelligent Home, which will also be available on video tape and CD-ROM. He may be reached at (405) 330-0718.

SOURCES

Dual-coax systems:

Molex, Inc.
2222 Wellington Ct.
Lisle, IL 60532
(708) 527-4238
Fax: (708) 512-8639

USTec
470 Pearl St.
Canadaigua, NY 14424
(716) 396-9680
Fax: (716) 385-6627

CCTV cameras, modulators, infrared repeaters, doorbell intercoms, and infrared remotes for computers:

Home Automation Labs
105 Hembree Park Dr., Ste. II



Roswell, GA 30076
(404) 442-0240
Fax: (404) 410-1122

Home Control Concepts
9520-108 Padgett St.
San Diego, CA 92126
(619) 693-8887
Fax: (619) 693-8892

Home Automation Systems
151 Kalmus Dr., Ste. M6
Costa Mesa, CA 92626
(714) 708-0610
Fax: (714) 708-0614

Home Automation and Security
286 Ridgdale Ave.
East Hanover, NJ 07936
(201) 887-1117
Fax: (201) 887-5170

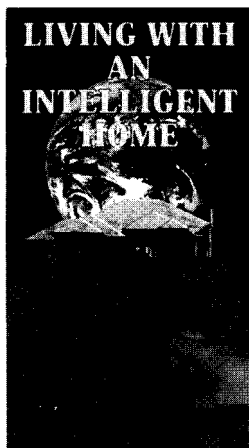
Video and VGA-NTSC adapters:
Your local computer store.

IRS

419 Very Useful
420 Moderately Useful
421 Not Useful

Let's Work Together.

Networking provides access to a world of resources, and Home Systems Network offers a world of resources to those who are interested in home automation. Check it out.



- ◆ **Are you looking for information?**
Obtain unbiased information about how to install and use all types of home automation systems from our books and *Intelligent Home* video tape series.
- **Are you looking to identify sources?**
Call our toll free number for a list of sources for any type of home automation dealers, products, or services.
- ◆ **Are you looking for marketing assistance?**
List your products and services in the Home Systems Network database and let us tell the world about them through our books, video tapes, television shows and referral services.

HOME SYSTEMS NETWORK P.O. BOX 3006 EDMOND, OK 73083 (800) 808-0718



got an interesting letter last week from a long-time HCS II user. He started out by telling me how he had added X-10 control of his pool pump,

attic fan, house ventilation system, and most lights. He had motion detectors, float-level sensors, door contacts, and so on connected as HCS inputs. He had the ADC registering various temperatures and even monitoring the battery-charging circuit. Like the real Circuit Cellar, there wasn't much you could do around there that would go undetected.

He even joked about the fact that he almost wrote me once to complain about the HCS's "bad something." Apparently, he was monitoring and recording the run time on his oil burner and comparing usage and deliveries to compute the cost and performance. The HCS record suggested that he was using about 200 gallons per hour during one week!

In truth, I would expect someone to write a letter to me about an HCS that logically concluded that heating his house costs more than the average shopping mall.

Realizing too that this was a little strange, he analyzed the options. Was there a leak? Was the oil man delivering less than the invoice record? Were these someone else's delivery bills? Where's the oil going?

Logically speaking, burning 200 gallons in one hour would cause a house meltdown. Forget that. Even at the correct rate of 1.7 gal/h, surely he'd notice an oil burner that ran 118 hours out of a 168 hour week. The only logical conclusion was that either the delivery and consumption records were flawed or some oil was not making it to the burner.

It's amazing what you can do with an HCS at hand. Undaunted, he simply changed the XPRESS code for a few of the things he already had on the system. He reaimed an outside motion-controlled flood light (with the flood light unscrewed) to point at the oil-tank fill pipe. With a piece of bubble gum stuck over the daylight

HCS Hard-wire Control: Back to Basics

sensor, the detector now worked day or night, sending an X-10 code whenever it sensed motion. If he was home, the HCS rang a chime so he could go watch. If he was out, the HCS turned on the power to his camcorder, which was aimed through a window at the filler pipe.

To make a long story short, the oil was being delivered properly and his oil burner was running at normal consumption. The loss was eventually attributed to a disgruntled neighbor who was using a 40-foot hose to siphon the oil into his own tank. Gotcha!

So much for the Sleuth 101 class. My reason for describing this caper is not to point out new uses for the HCS so much as it is to demonstrate the knowledge level of the user involved.

Applying real home control takes brains. Unfortunately, within an expanding universe of knowledge, it is impossible for any one person to know every thing. Especially when it comes to computers and applications, expertise is becoming quite diverse, and some of us are just hardware types forever.

The real danger comes from guys like me who know a little about a lot of things. When I am in charge of determining the level of documentation for complicated systems, sometimes I have a tendency to ignore being explicit about simple technical facts that seem so obvious that "surely the user must already know."

Considering the expertise of the guy I just described, especially using his HCS to catch a thief, I was floored when at the end of his correspondence he asked me why a siren and 12-1 flashing-xenon light he bought from Radio

STEVE CIARCIA

The HCS II accomplishes lots of goals—ease of use, energy savings, security, and automation. But, you've got to know how to make connections to real-world devices for it to be effective. For this, Steve takes us back to the basics, reminding us of how to do fundamental hard-wired connections.



Shack didn't seem to work when connected to the BUF-Term outputs.

Obviously, in our determination to protect the TTL I/O of the supervisory controller, we hadn't been explicit enough about the use of direct drivers and huffered receivers. Virtually all of his output controls were X-10; he hadn't used any direct-wired outputs. To

ask why a BUF-Term output can't directly control a high-current device like a siren implies that he views these lines to be in the same category as an X-10 control. Rut, they are quite the opposite.

I can only presume that a mere statement of specification is not enough for some. People learn to use hardware best through example and application. Only through concrete example do you learn the nuances of inductive loads, snubber networks, transient suppression, line losses, peak currents, and so on. Because nobody can be an expert in everything, many readers may not have experience in directly controlling real-world devices. I shouldn't presume anything.

In an attempt to fill the void, I'm going back to basics. From the HCS's digital I/O connector out, I'll try to explain the ramifications of real-world connection. Also, because I have received many inquiries about adding a watchdog timer to the HCS, I'll address how to do that too.

INTERFACING 101

With no expansion boards added, the basic HCS II and the HCS II-DX configuration has three forms of onboard control I/O: 8-channel ADC, RS-485 network, and 24 bits of parallel I/O. The parallel I/O comes from an 82C55 PPI configured for 16 inputs and 8 outputs.

CMOS TTL is virtually never used by itself as an external-control connection because it is extremely susceptible to out-of-spec voltages and transients. CMOS TTL has a basic voltage range of 0-5 V and is only guaranteed to drive about 1 mA of load. In addition, any voltage greater than 7 V or less than -0.7 V is usually quite lethal.

Unless these ports are used to connect to other TTL-compatible logic, they need special protection and amplification when connected to external real-world devices like

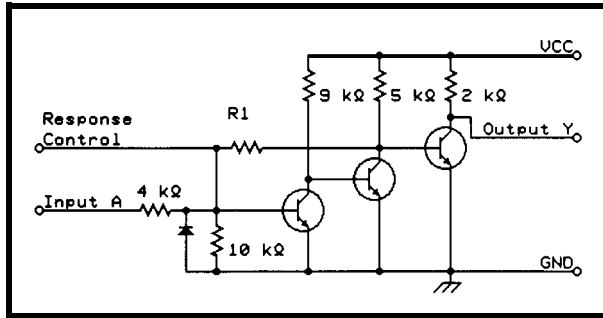


Figure 1: The BUF-Term board uses a common KS-232 receiver (shown schematically) to protect the HCS inputs to ± 30 V.

pumps, bells, indicator lights, door contacts, and switches. The function of the BUF-Term board and other HCS parallel interface boards is to provide that combination of protection and power.

The term used when providing protection to an input is called *buffering*. On the BLF-Term board, we use a common RS-232-to-TTL receiver as the input buffer. Shown schematically in Figure 1, the MC1489 (SN75189A) chip can withstand an input range of ± 30 V while converting it to a 0-5-V, TTL-compatible output. Considering the discrete resistors, diodes, and transistors required to devise similar security, the 1489 provides considerable protection at nominal cost. For an HCS operating on 12 V, these buffers easily protect the system from reversed connections, shorts, and so on.

The expression *buffered inputs* typically implies voltage-activated inputs. As such, dry-contact closures such as motion detectors, switches, and door contacts, which produce no voltage, won't work directly. This extra feature is effected by adding a pull-up resistor on each input to force all open inputs on. When a contact closure across one of these inputs

closes, it forces that output off. The BUF-Term inputs can therefore accommodate both contact closures and wide-range voltage inputs.

Since virtually all the input sensors, switches, and devices used on the HCS employ isolated contact closures, there is little concern that input control interfaces like the BUF-Term are common grounded. For commercial applications or assorted discrete voltage inputs not sharing a common ground, a 24-channel, optoisolated input interface called the *IDI-24* is available. Designed primarily for commercial use with SpectraSense 2000 (see sidebar for details), the IDI-24 requires a BUF50 I/O-expansion adapter to use it with the HCS 11 or HCS II-DX. Since noncommercial applications rarely need isolated inputs, I won't discuss them further here.

OK, now you know not to use TTL by itself, and you know that the input buffering on the BUF-Term offers both voltage protection and a current source for contact-closure inputs. With the exception of people who make three turns around an arc welder between the door contact switch and the BUF-Term input, these connections are remarkably foolproof.

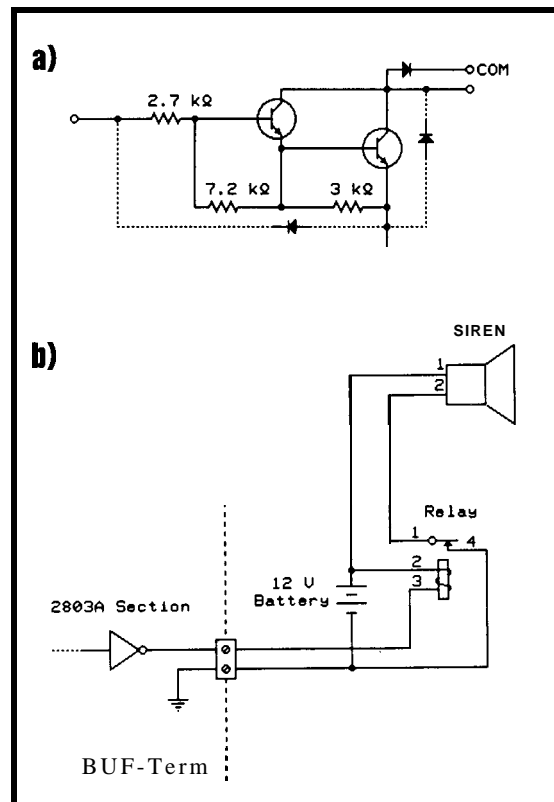


Figure 2: The DC drivers inside the 2803A (a) used on the BUF-Term board facilitate connection of external relays and indicators (b).



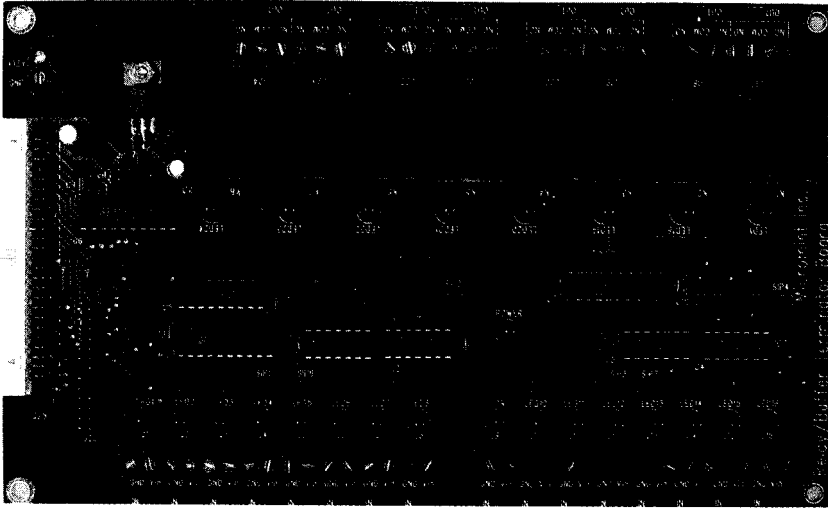


Photo 1: As an alternative to the BUF-Term, HCS users can now choose to use the Relay BUF-Term.

OUTPUTS

Many HCS users, who want to control things, primarily think of

lights and appliances connected via X-10 modules. Yes, the HCS does support both X-10 transmission and reception, but it also

controls devices via direct, hard-wired outputs as well.

When connected to a BUF-Term board, the HCS's CMOS TTL output port is converted into right 50-V-compatible, DC-driver outputs. Depending on the currents involved, these drivers can power indicator lights, beepers, small motors, and relays.

Whether you use hard-wired connections or X-10 is ultimately an issue of control reliability. Whether you can use the BUF-Term's driver outputs directly or add large relays depends on how much current is being switched. How you negotiate between these issues depends on your control methodology.

In XPRESS, the HCS can employ either open- or closed-loop control methodology. In an open-loop system, you merely issue a control command and presume that it happens. In a closed-loop

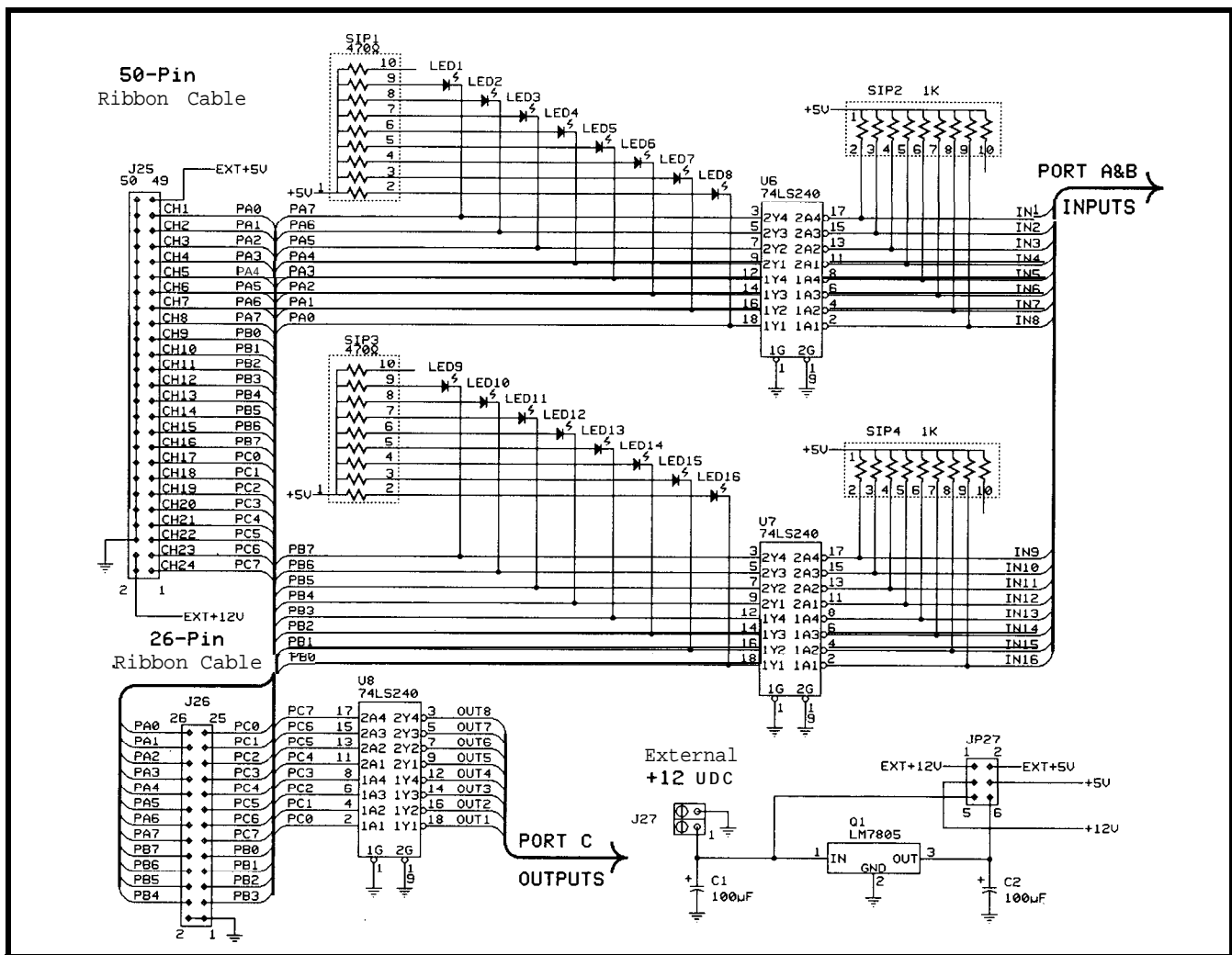


Figure 3a: The Relay BUF-Term board includes both a 26-pin interface for use on the original HCS and a 50-pin interface that matches the one used on most solid-state relay module backplanes.



system, you issue the command and a return signal verifies the consequences of carrying out that command.

Of course, there is no black-and-white dividing line for choosing between methodologies. Each control case requires a value judgment. The real issue is the robustness of the control connection and the importance of the control event.

If you just want to turn on a table lamp when someone enters a room, a simple open-loop command triggered from a motion detector suffices. Using an X-10 lamp module makes the control output both easy and wireless.

The downside to X-10 is that it is neither robust nor 100% dependable. We probably wouldn't care if an X-10 controlled light came on accidentally or not at all-a situation that can occur. In my opinion, this is why X-10, when used by itself, should be reserved for noncritical situations.

Eventually, you'll want to use your HCS to control something important. If instead of a motion detector and table lamp our control involves a pump motor and a pair of float switches, the consequences could be different.

First, you *cannot* presume when you send an X-10 command that it actually gets there! X-10 transmissions are easily trampled by vacuum cleaners, oil-burner transformers, fluorescent lights, and other high-EMI generators. Just because you command appliance module E5 on doesn't mean the pump attached to it actually will turn on. An even worse condition might be that the on command worked, but the off command didn't. Leaving a dry pump running is not desirable.

Even closing the loop by sending back a "pump on or off" signal only makes X-10 slightly more dependable. Interference that inhibited the initial command may persist. Sometimes X-10 codes just don't get there.

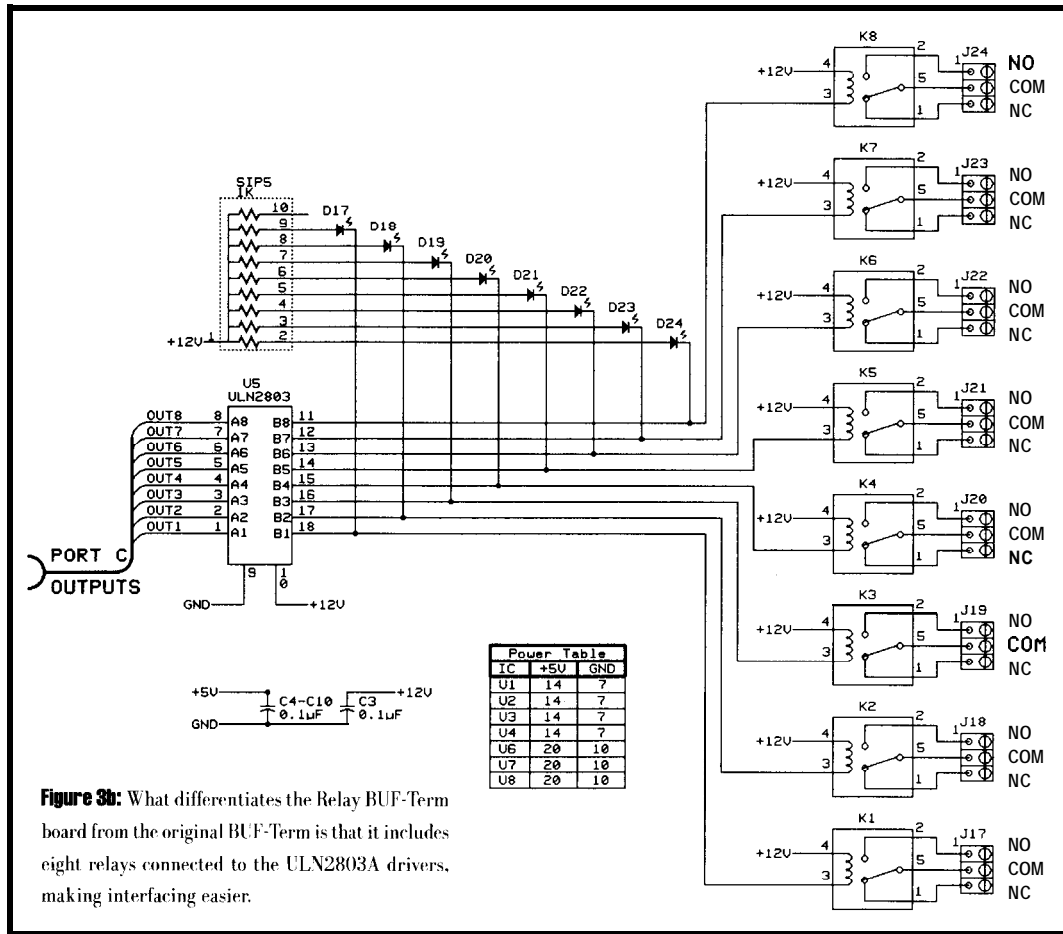


Figure 3b: What differentiates the Relay BUF-Term board from the original BUF-Term is that it includes eight relays connected to the ULN2803A drivers, making interfacing easier.

For important outputs, direct hard-wired control is the preferred method. X-10 is inexpensive, but a hard-wired relay directly controlling the pump is a surer connection. Because a hard-wired connection is also more robust, a closed-loop on/off confirmation is less needed since thrrr

is virtually 100% activation surety. Using direct-wired control, the pump should turn on and off infallibly.

Once you've made the decision to use direct hard-wire control, the only issue is the interface connection itself. The BUF-Term gives you eight

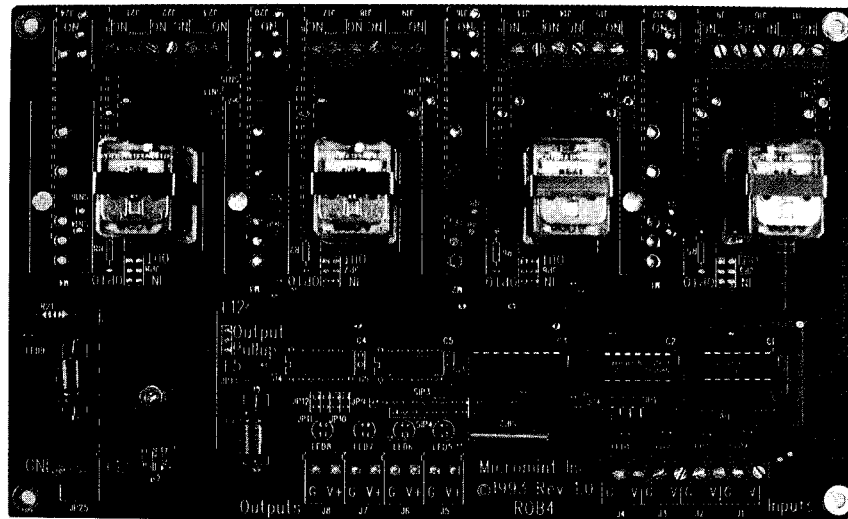


Photo 2: The ROB4 board high-current AC/DC HCS relay-adaptor board can be used in applications that require a more robust interface.

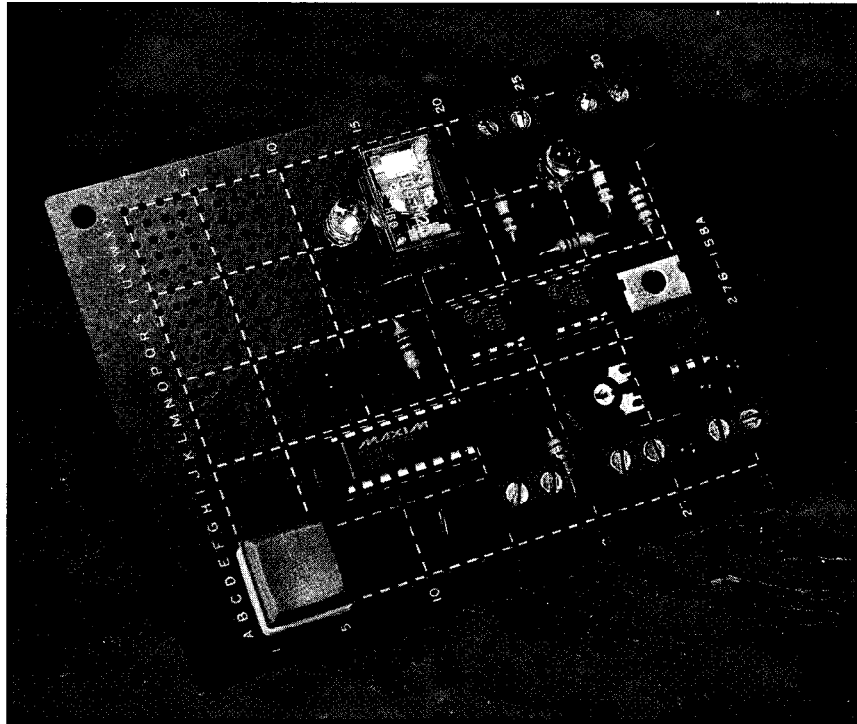


Photo 3: The watchdog circuit shown in Figure 6 fits neatly on a small piece of perfboard.

outputs. These outputs are conditioned through the UNL2803A Darlington array driver shown in Figure 2. Each output can sink 175 mA at 50 V when all are in use. You can sink 500 mA if only one is being used.

Please note that I have referred to these drivers as *sinking*, not *sourcing*, current. The UNL2803A has an open-collector inverted output. In XPRESS, when you command output 5 on, that output line is pulled to ground through the driver. The load (indicator light, relay, buzzer, etc.) is connected between the positive side of a common source voltage and the driver's output line (each output can have a different source voltage provided they all share a common ground). When the driver turns on, it acts like a switch on the bottom of the load, completing the circuit path to ground. The only limitation, as I said, is how much current can be pulled to ground.

The siren and xenon flasher mentioned earlier take 3 A and 1 A at 12 V, respectively. Since they are well above the peak 500-mA driver rating, it should be easily understood why neither can be connected directly to the BUF-Term.

To accommodate greater load currents, we add relays. The BUF-

Term output can be used to drive a relay coil, and that relay's contacts control the load. The typical, low-cost relay has a rating of 3 A at 28



VDC and 2 A at 240 VAC (resistive load) with coil currents of 40-60 mA. Controlling the siren is merely a matter of adding a small relay. Controlling a water pump requires a larger relay, perhaps 8-10 A. As long as the coil current fits within the 2803A drive spec, there is no difference in the connection. Note that higher coil voltages require less current. If you draw too much current with a 5-V coil, switch to a 12-V device.

PACKAGED SOLUTIONS

I don't want you to think we left you entirely on your own. The configuration of the HCS and BUF-Term is meant to be economical. The BUF-Term was designed so that you could easily add an external relay when required. However, if you ultimately have to add eight relays for your application, you might be asking why we just didn't do it in the first place.

New products from **PARKS ASSOCIATES**

the organizers of Habitech, The Home Systems Trade and Training Show

The Business of Home Automation

Know the market!

The Business of Home Automation will address the market on home systems for those who plan on becoming a dealer/installer, or are already a manufacturer or home builder.

Applications and markets.

Some of the systems covered in this report are HVAC, advanced controllers, entertainment, in-home networks, basic controllers, and lighting.

X-10 LTD. Myth & Reality

Comprehend the structure!

X-10 technology has been around so long, many think they know all of its capabilities. For the first time, a comprehensive report on X-10 Ltd. is available, explaining not only its corporate structure, but also its technology.

Call for special pre-published prices. (800) 727-5711

Coming in 1995 . . .

Habitech95

The Home Systems Trade & Training Show

Mark your calendars now!

The third annual Habitech, Trade & Training Show will be held May 17-20, 1995 in Atlanta, Georgia. Contact Judy Brendemuhl at (800) 727-5711 for more information.

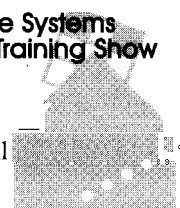


Figure 3c: The input section of Relay BUF-Term board uses the ubiquitous 1489 RS-232 receiver chip to provide a ± 30 -V input range.

Actually, we do (see Photo 1). Figure 3 is the schematic of the Relay BUF-Term board. It connects to the HCS's PPI the same way as the straight BUF-Term and has 16 buffered inputs. Instead of open-collector drivers, this board has eight SPDT relays with the ratings I've stated. A siren or xenon flash is easily accommodated by the 3-A contacts. An additional advantage of the Relay BCF-Term is that all inputs and outputs have LED status indicators.

While the BIJF-Term is more economical, the Relay BUF-Term offers easier **output control**. Mechanical relay contacts don't care about polarity or whether you are switching AC or DC current. If our oil-burner guy had the Relay BUF-Term, his siren and flasher would have been easily accommodated, even though he wouldn't have known exactly why. There is also another alternative if you only need a few relays. Especially when you

need more than 2 A, you can use the ROB4 adapter board (see Photo 2). Shown schematically in Figure 4, the ROB4 is a 4-channel, high-power relay adapter board. The ROB4's relays are rated for 10 A at 240 VAC. Each relay is individually controlled by a BLF-Term driver output. The ROB4 also provides switch-selectable polarity and LED status indicators.

HCS-CONTROLLED AUDIO

After making the case for HCS-controlled relays based on their improved reliability over X-10 in critical situations, perhaps a more enlightening example might make a better recommendation.

I was in one of those audiophile shops recently ordering something.

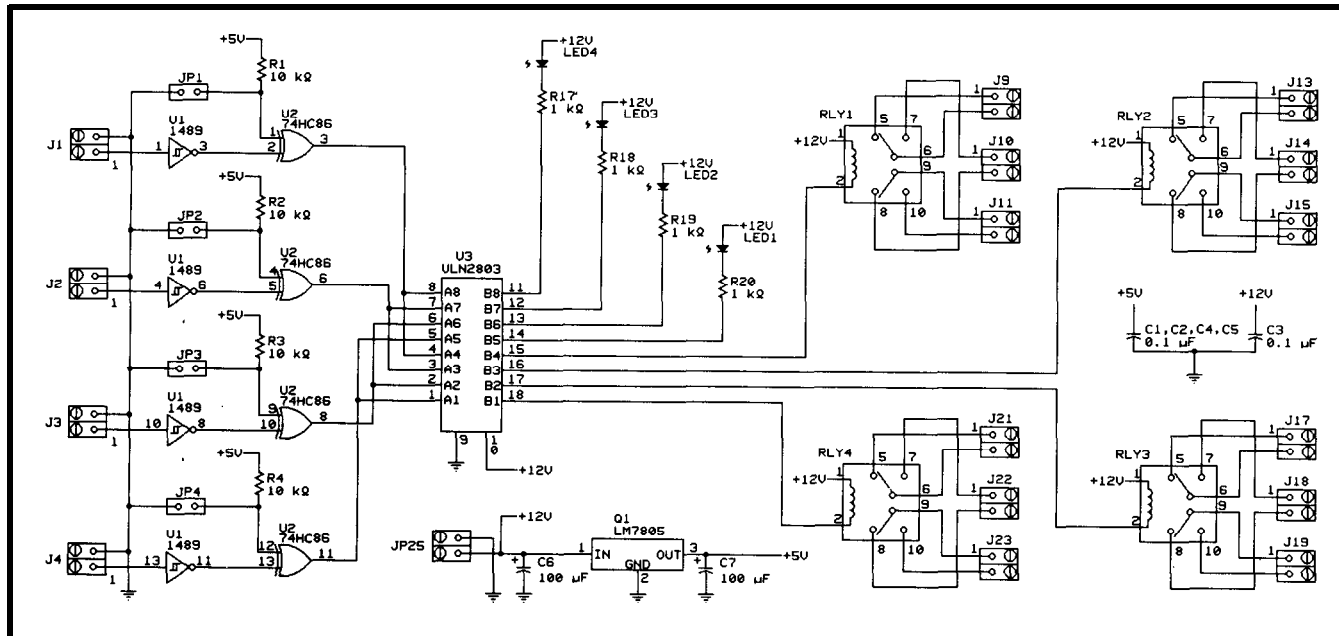
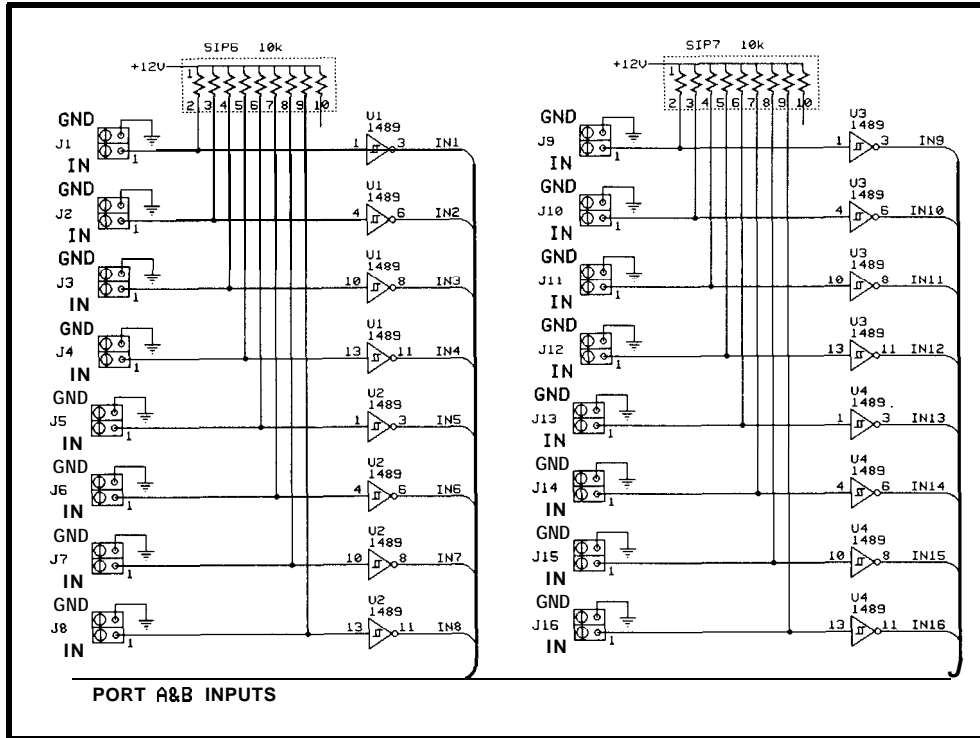


Figure 4: The ROB4 high-current relay adapter can be used in applications that require higher switching currents.

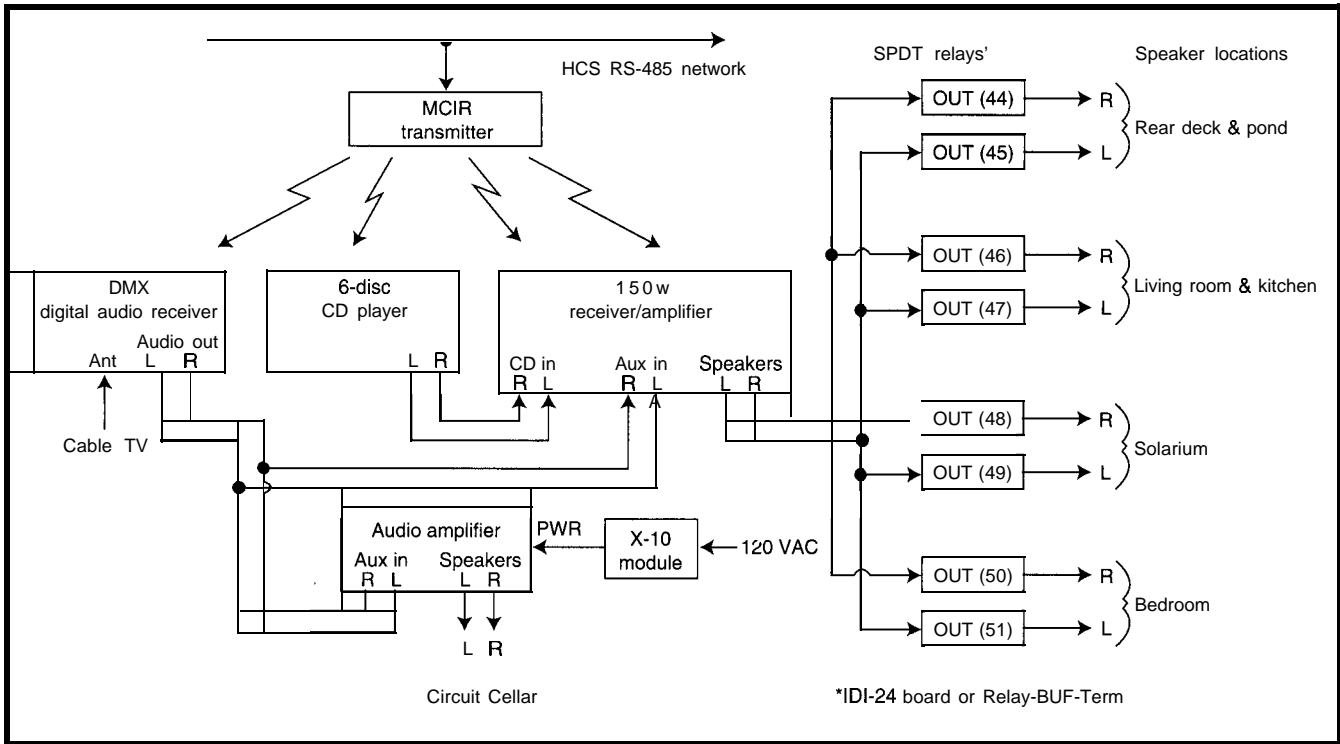


Figure 5: The Circuit Cellar HCS-controlled audio system uses the HCS as the brains to automatically switch speakers on and off in occupied rooms. The HCS can also select the audio source and adjust the volume remotely.

During the conversation, they mentioned installing a whole-house stereo-speaker switching system as part of a large home-theater installation. The switcher was an off-the-shelf commercial unit, which used wall-

mounted, push-button panels in various rooms. The panels activate the system, make the audio selection, and set volume. It definitely sounded like a fine installation, but I nearly choked when they said it cost \$14,000.

Wait a minute. Play a little music? Switch a few speakers? Fourteen grand? No way! Hang a few relays on the HCS and I'll bet we can do it for a couple hundred easy.

Well, don't hold me to the price, but adding automated audio control to your HCS

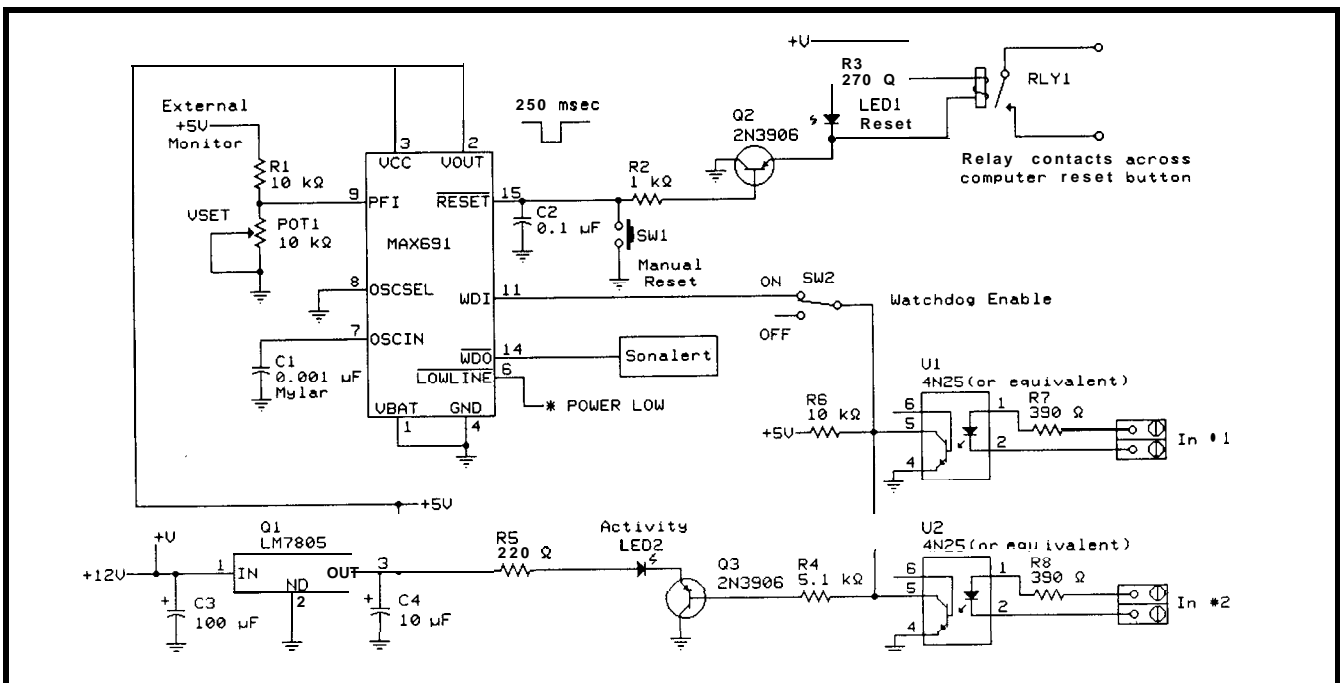


Figure 6: The MAX691-based watchdog timer circuit has a heartbeat period set for 6 s.



is a relatively trivial hardware exercise and costs considerable less than \$14,000.

The humorous consequence of installing such a system is revealing, however. I can save \$1000 a year on computerized HVAC control. call the HCS and find out when the last car came in the driveway or make the house look completely lived in while we're away. All this description goes in one car and out the other with most people. If I mention that the stereo follows me from room to room, it's suddenly like that E.F. Hutton commercial. Everybody perks up and listens. Perhaps I've been concentrating on the wrong end of home control.

If only for a short time, perhaps now we have everyone's attention. Figure 5 is the block diagram of the Circuit Cellar HCS-controlled speaker-switching system. I installed dedicated audio components for this system which are not shared with other

listing 1: This simple XPRSS1 program toggles an output bit about once per second

```

If Timer(0)=off then
  Timer(0)=on
End

If HCS_Heartbeat=off AND Timer(0)>=1 then
  HCS_Heartbeat=on; Timer(0)=on
End

If HCS_Heartbeat=on AND Timer(0)>=1 then
  HCS_Heartbeat=off; Timer(0)=on
End

```

functions. A dedicated system has the benefit of always having a predictable configuration (nobody messing with the knobs).

It basically consists of a 150-W stereo receiver which is connected through four pairs of SPDT relay (one for each left and right channel) to speakers throughout the

house and on the outside deck (more speakers could be added, but I didn't have any reason to). The relay board is an IDO-2424-relay expansion board. I used this large relay board because I also switch a number of sirens and sounding devices through it as well. You could use the eight re-

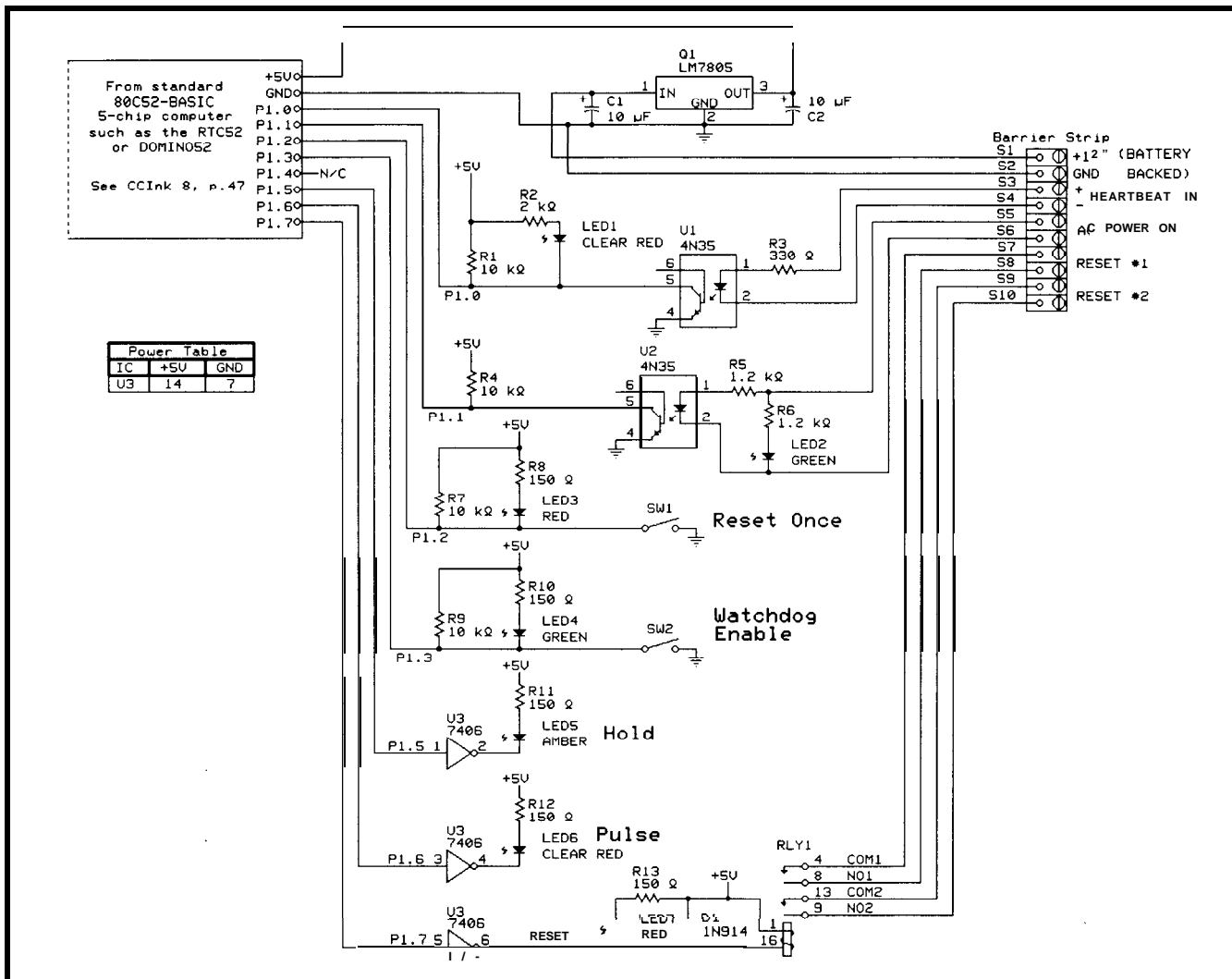


Figure 7: The intelligent microcontroller-based programmable-function watchdog timer takes such factors as AC power and target activity into account.

lays on a RelayBUF-Term board if you were just switching audio.

Audio sources come from either a 6-disc CD player, integral FM tuner, or a 30-channel, cable-company supplied, digital-audio receiver called *Digital Music Express* (DMX). The audio sources and the receiver can all be infrared remote controlled.

Operation is straightforward. An MCIR-Link (the IICS infrared remote-control synthesizer) is trained with the codes for the DMX converter, CD player, and FM tuner. To use the system, the HCS turns on the receiver and sources, sets an input channel (CD, DMX, or FM), sets the volume, and selects a speaker. Since I have motion detectors everywhere, speakers can easily be selected wherever motion is detected. You can just leave them on after that or switch off a room if no motion is sensed for half an hour or so. Alternatively, you could use infrared receivers in other areas and control the system manually or have the HCS set up listening areas.

I specifically chose an amplifier that could handle a low impedance of 2 . While the long connecting wires offer some resistive protection, it is possible to have four sets of speakers on at one time and a marginal amplifier might clip the signal. Since I don't even detect an audio-level change when any additional speaker sets kick in, the amplifier apparently has more than enough headroom.

I use a separate amplifier and speaker set for the Circuit Cellar itself. In the relay-switched system, all speakers are at the same volume. Because of the layout of my contemporary-styled home, this is in fact advantageous. However, with the Circuit Cellar being audio isolated from the rest of the house, the easiest way to have independent volume control is to use a separate amp. An X-10 appliance module turns the amplifier on and off.

Finally, having an HCS-controlled audio system offers a new dimension to a security system. Consider queuing sound effects or a kodo drums CD the next time someone pulls into the driveway when the alarm is set!

SpectraSense 2000

The SpectraSense 2000 is an all-in-one home and light-industrial control system designed for easy installation and maintenance. Four or five of the more popular IICS components were combined onto a single board and put into a heavy-duty steel enclosure to take the guesswork out of connecting together separate subsystems. This programmable building control system manages household circuits, appliances, and HVAC systems either directly or remotely. The system can be integrated with building control and security systems such as HVAC controls, automated valves and dampers, security systems and alarms, multimedia entertainment systems, and virtually any other building-automation product.

The SpectraSense 2000 offers a wide variety of advanced control features. You can:

- use the board without a dedicated PC connection
- directly connect up to 24 contact closure-type inputs
- directly connect up to 24 buffered outputs
- directly connect up to 8 analog sensors (temperature, humidity, light level, etc.) using 8-, 10-, or 12-bit resolution and gains of 1, 2, 4, or 8
- enjoy 2-way X-10 power-line control
- generate, receive, and monitor telephone calls using a DTMF phone interface
- connect up to 31 network expansion modules over a single twisted pair up to 4000' away
- directly plug in other expansion modules such as voice output and additional digital I/O
- program with X-PRESS control language using a PC-compatible computer

HOME SYSTEMS NETWORK

releases *Intelligent Home*, a series of video tapes for home automation dealers and installers.

Living with an Intelligent Home introduces the concept of home automation and includes:

- a tour of an intelligent home
- a cost vs. savings analysis
- options and features available

Lighting Controls, Volume II, delves into the intricacies of lighting control. The topics covered:

- installation of various switches, receptacles, remote controls, and sensors
- wiring in existing home environments
- coping with multiple power standards

Upcoming videos in this series will be released every 30-60 days. They will present such topics as:

- security systems
- entertainment and communications
- energy management
- windows, doors, and gates
- plumbing and outdoor systems

Home Systems Network

P.O. Box 3006

Edmond, OK 73083

Tel (800) 808-0718



WATCHDOG TIMERS

An HCS installation is a classic case of "hurry up and slow down." The HCS operates at a million instructions per second, analyzing a static situation *only* to conclude it should logically do nothing. Until an event occurs or the static condition changes, not much happens. Of course, when that event occurs, you want instant action. You don't want to find out that the system has gone south when control is most needed.

Automatically monitoring an electronic system for basic operating integrity can be as simple as an added test routine or as complex as NASA's triple-redundant hardware and logical arbitration. The degree of automatic testing is predicated on the complexity of the systems being controlled and the necessity for their control maintenance.

While we might use the HCS to mess with the HVAC a little, this hardly qualifies as life support. In a basic HCS installation, it is reasonable for us to conclude that if it executes properly for one routine, it's operating properly- for all (presuming good code). If we write a routine that maintains an observable, repetitious pattern, then if that pattern stops, we know something adverse must have happened. Frequently, simply resetting the machine is the solution.

We typically call the repeatable pattern a *heartbeat*. The circuit that monitors the regularity and timing of the beat is called a *watchdog*. Because we configure them primarily to look for any heartbeat within a measured time rather than a particular periodic waveform, *watchdog timer* is a more appropriate term.

Figure 6 shows a watchdog-timer circuit using a MAX691 (see Photo 3). While you could glue it directly on the HCS processor board (without the relay and isolators), I chose to assemble the unit as an offboard watchdog. The watchdog function is enabled by activity on pin 11. If left open or high, the watchdog does nothing. The capacitor on pin 7 sets the watchdog period. Here, the period is set for 6 seconds.

Within XPRESS, you can write a simple independent program that blinks an output LED about once **per** second (see Listing 1). The heartbeat is one of the BUF-Term outputs. As long as something happens on that line within 6 s, the '691 will clear and watch again. If it times out, the relay closes causing an HCS reset.

The reset relay contacts are best connected across the reset push button or reset

Listing 2: The BASIC code for the watch kennel uses print statements for testing purposes only.

```
100 REM Watchdog Time- Rev. 1.0
105 REM Bit0 Pulse in, Bit1 Power Monitor
110 REM Bit2 Reset Once Switch, Bit3 Enable Switch
115 REM Bit4 N/A, Bit5 Hold Indicator
120 REM Bit6 Pulse Indicator, Bit7 Reset Relay
130 REM
200 REM Setup constants
205 N=30: REM N=Timeout in seconds
210 VAL=0
220 REM
225 REM
300 REM Check Bit3 Enable switch
305 A=PORT1 : REM Read switch input bit
310 PORT1=31: B=A.AND.8
320 IF B=8 THEN PRINT"Switch off" :GOTO 305
325 REM Continue if enabled
327 PRINT"Switch on"
330 REM
400 CLOCK1
410 Time=0: REM Start second timer
420 A=PORT1: REM Read heartbeat input
425 PORT1=95 : REM Turn on port1 bit6
430 PULSE=A.AND.1; SW1=A.AND.8
435 IF PULSE=VAL THEN GOTO 460
437 IF PULSE<>VAL THEN VAL=PULSE
440 IF SW1=8 THEN GOTO 305
445 IF TIME>=N THEN GOTO 500
447 PORT1=31: PRINT" Pulse"
450 GOTO 410
455 REM
460 PORT1=31: PRINT"No Pulse", TIME
465 IF SW1=8 THEN GOTO 305
470 IF TIME>=N THEN GOTO 500
480 GOTO 420
490 REM
500 REM Watchdog Timeout
510 PRINT "Timeout"
520 A=PORT1
522 C=A.AND.2
525 IF C=2 THEN GOTO 305
530 TIME=0 : REM Reset pulse for 3 seconds
540 PORT1=255: PRINT"Reset Pulse on"
550 IF TIME<3 THEN GOTO 540
560 PORT1=159: PRINT"Reset Pulse off"
570 TIME=0
580 REM
600 A=PORT1: C=A.AND.4
605 IF C=0 THEN PRINT"Holding": PORT1=63: GOTO 600
610 GOTO 305
```

header on the HCS. Because there are buffers between the processor reset pin and the reset line on the expansion bus, attaching the relay contacts there does not reset the system unless these buffers are removed.

If the circuit in Figure 6 is a watchdog, then the circuit in Figure 7 is a watch kennel (see Photo 4). It does the same basic function-waits for activity and pounds on reset-but it offers significantly more programmability and user-controlled options.

First, this unit is not a single chip.

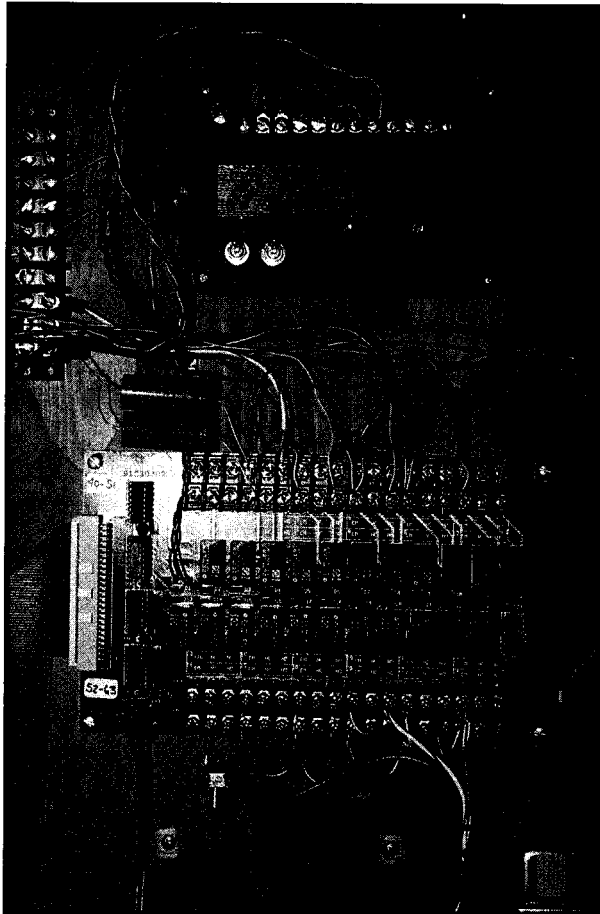
Instead, it is a complete RTC52 80C52-BASIC computer pro-

grammed to synthesize '691 functions.

Without arguing how to protect the computer that's protecting the computer, let's just say that it's battery-backed, ruggedized, and has its own watchdog. The reason I went to the more elaborate circuit is that as I expand my HCS system, timing becomes relative to the workload. If I use a network I/O module to provide the heartbeat, then the programming is executing through the processor, through the network communication



Photo 4: The watch kennel (top) and the IDO-24-based speaker switches (bottom) are all connected to the same HCS that runs the rest of the house.



lines, and then through the link itself. You could call it a compound check. Once you involve the network, however, you have to contend with delays as other links are serviced.

One of the other compounding considerations in a now expanded HCS system is the effect of power outages. While my system is battery-backed throughout, not everything controlled is. There are two sets of reset contacts so one set can go to other systems that also need resetting. During a power outage, I hold off the watchdog function. After power returns and is stable, I hit reset to synchronize everything again.

Listing 2 contains the BASIC code for my watch kennel. The PRINT statements are simply for testing. All the LEDs simply let this watchdog know the system is working.

IN CONCLUSION

For some people, the basics are boring. For others, they are a revelation. For the average HCS owner, I present this article as nothing more than a reminder that home control involves physical connections at some point and the HCS has a considerable number of options available. Mastery of the subject then is merely a blend of determination and understanding.

Steve Ciarcia is an electro71c.s engineer and computer consultant with experience in process control, digital design, and product development. He may be reached at steve.ciarcia@circellar.com.

SOURCE

The following are available from

Circuit Cellar, Inc.
4 Park St.
Vernon, CT 06066
(203) 875-2751
Fax: (203) 872-2204
Internet: sales@circellar.com

- ROB4-4R6X:** Industrial high-current relay output board \$199
- HCS-RBUF-1K:** HCS relay buffer board kit with 16 input buffers and 8 output relays \$169
- IDF-24:** 24 input optoisolators for the HCS \$239
- IDO-24:** 24 output relays for the HCS 5349

IRS

- 422 Very Useful
- 423 Moderately Useful
- 424 Not Useful

At Home Control Concepts we know you'll just love us.

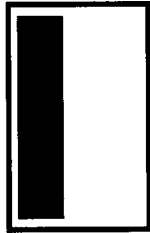


All our customers do.

Harley Davidson * Microsoft * Omni Interactive * IBM * Commercial Electronics * Disney Land * Disney World * Tandem Computers * Dept. of the Navy
 American Aerospace Corp. * Morpheus Lights * U.S. Dept. of Energy * Stratagene * MTS Inc. * Panavision * Georgetown University Law Center
 Novell * AVA Design Group * Federal Communications Commission * Lutron * Motion Cameras Co. * Stanley Tools * Teledyne Brown Engineering
 CBS Television * KTVO * WINND Radio * WSLA TV * KMEG TV * WLEX TV * Good Morning America * Honeywell * Aquadine * Texas Instruments
 Martin Marieta * Dept. of Justice * Alaska Governors Office * City of Ashville * Hewlett Packard * Wang Laboratories * Intelligent Home Systems
 Toyota * Enabling Technologies * CBS News * Smart Homes (several) * Datavision * Laserbeam Creations * Microbit * Museum of Nebraska History
 Fire Station #29 * Electronic Interiors * PerformaHome * New York Funding * Westcom Inc. * Illuminations * Amana Refrigeration Inc. * Safe N Sound
 Loshooton County Sheriff Dept. * Disney Imagineering * Walt Disney Engineering * Gillman Builders * IntelliHome * Bell Canada * Delta Airlines
 Mitsubishi Electric * Omniplex Science Museum * United Methodist Church * Neil Square Foundation * Gateway State Bank * Alpha House * MDC Inc.
 Pennsylvania Power & Light * New Jersey Cable TV * NHC Welfare Office * City of Longview * Intracorp * Computer Applications Journal * Cybertec
 Tecktron * Geniehouse * Cal Global * Kholer Faucets & Hottubs Co. * Netmedia * Broadmoor Police Dept. * Milwaukee World Festival * University
 of Delaware * University of Ohio * Louisiana Tech University * Louisiana Rehab Hospital * Michigan State University * University of Birmingham
 Virginia Tech * Spokane School District * Elmhurst School District * New York University * State of Nebraska * Texas Tech University * CMC Oregon
 State * Oregon State University * College America * University of New Mexico * University of Maryland * John Hopkins Hospital * Montreal Childrens
 Hospital * Gillette Childrens Hospital * C.W. U. Medical Center * DMC Harper Hospital * Montreal Childrens Hospital * Eye Consultants * Rehab
 Dimensions * Plus Customers in Scotland, Columbia, Italy, Turkey, Singapore, Israel, Argentina, Japan, Mexico, Canada, United Arab Emirates,
 Australia, France, Jamaica, Virgin Islands, Cayman Islands, New Zealand, and other Customers Abroad.

Free Wholesale Catalog!

Call 24 Hrs. **1-800-422-4024**



It didn't happen all at once. No. That would have been too easy. It had to take months to gradually manifest and several more weeks for me to become aware

that something wasn't right. I should have seen it coming. The warning signs were all there: reduced output, lack of stability, even the inputs seemed wrong.

Then one black day. I finally realized what had happened. The design had seized up. Stuck, stopped, halted. No, I'm not talking about hardware or software—the design itself was hung. Or, to be more precise, the designer was stuck. Me, the self-proclaimed expert of elegance, was defeated, done in by a common foe: creeping featurism.

I learned an important lesson that day. When the same person does both the specification of features and the design of a project, a runaway feedback loop can develop that results in more features than it is possible to cram into the design. It's the old "it's hard to say no to yourself" routine, and it goes something like this. You say to yourself, "Wouldn't it be neat if it did this?" And, then you answer, "Yeah, that would be cool!" Voilà, another feature is added.

We've all had a good laugh over marketing requirement documents, and routinely recite the 1001 ways to kill an unreasonable feature request. Hut, we're just not masochistic enough to use them on ourselves. Or, at least, I wasn't.

So, what did I do? I got help. I talked to other people about the various features and ranked them in order of desirability. Then I restarted the design, taking the features in order. It soon became clear where to draw the line.

You might have guessed by now that the design I'm talking about is a user interface. How many times have you heard someone complain about how hard it is to program a VCR? Hut, do they ever say how it could be improved? Whenever someone says

A Different Set of House Keys

Making the Most of a Small Keyboard

JEFF FISHER

Designing a hand-held IR remote for doing general-purpose home automation is no small feat. Jeff discusses the design issues involved, presents some options, and settles on a set of features for the "ultimate" remote.



Photo 1: The "ultimate" remote control allows not only manual control of in-home devices, but can be used to perform some programming as well.

that to me, I draw six buttons and a seven-segment display on a piece of paper. "Co ahead," I say, "Just how would you do it?"

THE PROBLEM

I had a similar problem on a much larger scale: if it's hard to make a VCR that's easy to program, imagine designing a user interface for something that controls an entire house!



Home automation is all about control—remote control, automated control, unified control. Beyond that, home automation is a very personal thing. What individuals want controlled and how they prefer to wield control is unique to each, person.

Some people prefer table-top buttons, others want wall-mounted switches, and a few like telephone interfaces. But, the most popular method of control is the wireless, hand-held remote control. (You may insert the appropriate gorilla grunts here. It's a myth, you know, that virility is directly related to the number of buttons on a remote control. Virility is related to how many things a remote can operate!)

I was designing the ultimate remote control (see Photo 1). I wanted users to be able to:

- use the remote control like a universal remote which controls a TV, stereo, VCR, and so on. (The remote control transmits RF to a console. The console actually does the infrared emitting.)
- send simple X-10 commands from the remote to control other appliances. (Again, the base actually transmits the X-10 commands.)
- execute complex programs with a single-button press
- choose which keys would perform which functions
- program complex functions through the remote. (I was trying to cater to the arm-chair programmer.)

How in the world could I cram all this into a small keyboard? Even with sophisticated users, it was difficult figuring out.

Sure, it's easier to write big hairy programs on a PC and download them to your whole-house controller. But, it turns out that most user needs are not complex—merely numerous and impossible to identify all at once. Why force the user to go boot up a PC, connect the controller, load the program, and so on when there are *theoretically* enough keys on the remote to do it from an easy chair? Besides, not everybody has a PC available at all times.

After consulting with my colleagues, we boiled down our needs into these requirements. We wanted:

- user-assignable keys which enable users to set the system up so that single-use keys would issue infrared commands (working

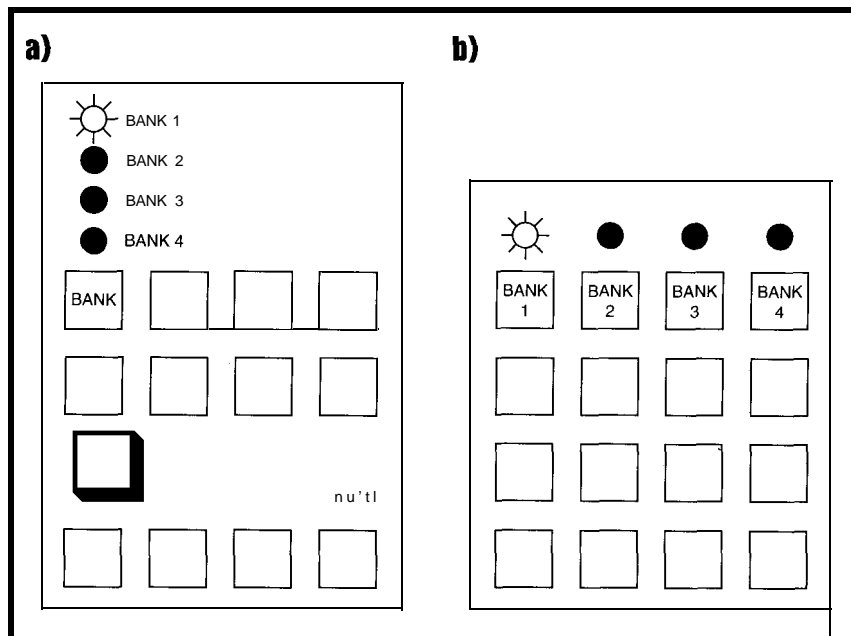


Figure 1: a) If you are primarily concerned with limiting the number of buttons on your remote, buttons can be programmed to offer different capabilities depending on which bank (or mode) they are in. b) As an alternative, each bank can have its own reserved key. While this has the advantage of single-button pushes to select a bank, it does require more keys. With either design, it is important to remember to use LEDs to indicate the current bank.

just like a universal remote control) or run their programs.

- punch-in numbers (we call them IDs) and the ability to select one of a dozen func-

tions to operate on these IDs. To do this, we needed a way to define the IDs using infrared codes, X-10 addresses, programs, or groups of other IDs.

- let the user define programs from the remote. This was the hardest part to design. The programs were written in Common Application Language (CAL), part of the CEBus specification, and we had identified over fifty functions that the user could access.
- the ability to run programs based on external events such as received X-10 codes and specified times.
- to allow multiple mappings of keys so that users could change their layout on the fly, and so multiple remote controls could have different mappings.

During the design, we tried a lot of different ideas, learned a lot about embedded-system user interfaces, and even more about the people that use—and complain about them.

If you are doing a project that has more functions than you want to provide individual keys for, here are a few ideas that may be helpful.

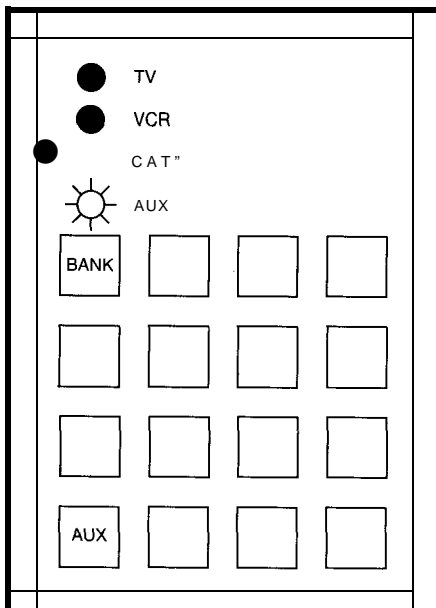


Figure 2: The Aux option provides a more complex set of keys for programming and other multikeystroke operations. Although the average user would find this netherworld intimidating, it offers the programmer a paradise of options.



HOW MANY KEYS?

You should have enough individual keys that the user can perform the most common operations with a minimum of keystrokes. This means a single keystroke for commands with no arguments. (Arguments are usually numbers such as times, addresses, etc.) If a command requires one argument, make sure that just the numbers and the command need to be entered. If a command takes multiple arguments, a single key should be used to separate the arguments.

Let's say that you identify six commonly used commands, and that each command can have zero or more numeric (decimal) arguments. You need a keyboard with ten digit keys, six command keys, an enter key to separate arguments, and probably a cancel key—that's at least eighteen keys. We'll look later at a more accurate way to estimate the number of keys you'll need.

But, the keyboard does not have to have a separate key for each function. Calculator manufacturers have

been doubling up keys for years. Designing such a user interface may seem simple—if you've never done it. Sure, the firmware is straightforward. But, designing the layout can fry your brain! I've been there.

The problem is, oddly enough, too much freedom. A sixteen-button keyboard has 65,536 possible four-keystroke operations! Before you start assigning keys, you must impose some order to the keyboard. We'll look at several tried and true way to get a lot more functions out of each key, while still imposing enough order that all functions remain accessible.

But, back to the basics. We were designing a remote control for a whole-house controller. A remote control is nothing more than a wireless keyboard—the same problems apply. Much of the time, this keyboard (even though it transmits RF) works like your infrared remote control. So, it needed enough keys to make a decent TV remote. In another mode, the user could enter arbitrarily complex commands into the keyboard. We needed numeric keys, 10-15 function keys, a few syntax keys, and some keys that would let us access more esoteric

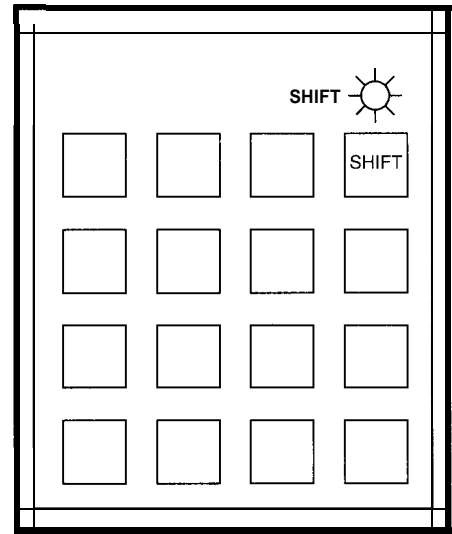


Figure 3: The shift key changes the next key pressed. It is essential to remember, however, that the shift key may only be appropriate for some banks, and you do have the option of a double-shift key.

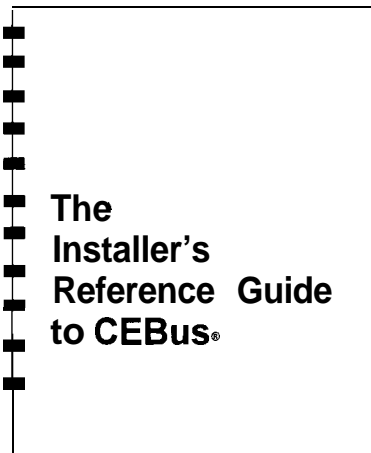
functions. Thirty-two is a nice round (albeit nerdy) number. Thirty-two keys also seemed adequate for an infrared remote control.

If you have more functions than you have keys, you can provide different modes

New
Information -
Special Price!
\$69⁹⁵

Hands-on info for CEBus automation.

Written by
CEBus expert
Grayson
Evans and
published
by Parks
Associates,
the manual
contains
easy-to-use
instructions,
including
graphics and
diagrams.



CEBus is a registered trademark of the
Electronic Industries Association

Uses the CEBus® standard!

This manual provides detailed instruction on the backbone wiring that will interconnect the electronic home of the 90s.

For installers of all types, and all applications. Emphasizes CEBus and its application for security, entertainment, lighting, telecommunications, and energy management. Designed for on-site use, with clear, easy-to-use instructions, including graphics and diagrams. It reveals "insider" information on how to wire for current and future automation products and services.

Available now!

The Installer's Reference Guide to
CEBus is \$69.95. To order, call

Parks Associates at (214) 490-1113,
or toll free (800) 727-5711.



**PARKS
ASSOCIATES**

or “banks” for the keys (see Figure 1a). Similar to the caps-lock key on a computer keyboard, you reserve one key that permanently switches the meaning of the remaining keys. Just as the caps-lock key has an associated LED, you should provide an indicator that shows the current bank setting. Alternatively, as Figure 1b shows, you can reserve one key to select each of the possible banks. This uses more keys, but ensures that other banks are never more than a single key-stroke away.

To simulate at least three infrared remote controllers, we needed three banks of simple single-function keys. The user could program each key to issue an infrared code or execute a stored program. Our fourth bank provided a more complex set of keys for programming and other multikey-stroke operations which we called the Aux bank. Rather than reserve four keys to select the bank, we used one key as a bank-witch key. Each press advances the current bank by one. Turning on one of four LEDs shows the current bank.

On this remote, users would likely leave the keyboard in one of the infrared simulator banks most of the time. If they wanted to perform some quick command in the Aux bank, they would:

1. Look at the bank indicator.
2. Press the bank key up to three times
3. Perform the operation
4. Change back to the original bank

So, one key is reserved in all banks that temporarily switches the remote to the Aux bank (see Figure 2). Aux bank commands are structured with a definite ending so the user can press the Aux key in any bank and type in one Aux bank command. Then, the keyboard reverts back to the previously selected bank.

This functionality has the additional advantage of making the documentation easier to write. Instead of having to write “Make sure the keyboard is in the Aux bank (see the procedure on page xx.), then press ‘1’ ‘On.’” you can write “Press ‘Aux’ ‘1’ ‘On.’” Mind you, documentation would also be simple if you used a separate key to select each bank.

Another way to get double duty out of each key is with a shift key (see Figure 3). However, the shift key may only be appropriate for some banks. On a standard keyboard, the shift key must be held down

Line Number	of...	Description	Count
1	Common function keys	Various	14
2	Digit keys	0 through 9	10
3	Syntax keys	Enter, Begin, End, Var	4
4	Shift keys	Shift	1
5	Keys common to all shift states	Cancel	1
6	Bank change keys	Bank	1
7	Keys common to all banks	Aux	1
Total			32

Table 1: A critical step in designing a remote is mapping the keys. The numbers in the Count column indicate the options available on the remote we designed.

while you press another key. This type of shift key is usually a little more expensive to implement and, on small key pads, is not usually ergonomically sound. Thus, the shift key on nonstandard keyboards is usually implemented as a separate key press. It seems that most people have grown used to this with calculators and similar devices.

You can also have double-shift keys, where the user presses the shift key twice before the function key. This gives more functionality while not using any extra keys. An alternative to the double-shift key is a second shift key. My calculator actually has three shift keys: brown, red, and white. Legends above the other keys are printed in these colors.

Often, an indication of the shift state is provided. If the shift key is pressed inadvertently, pressing it a second (or third) time cancels the shift. Note that the shift mode automatically turns off with the next key press. (A shift that stays active until canceled is more like bank selection described above.) The shift key(s) might only be operational in certain banks, so you don't have to give up these keys in other banks.

Because of a rich programming language, we ended up with more than 64 functions and numerals needed in the Aux bank (see Figure 1). We opted for the shift-and-double-shift-key method of accessing these functions. We put some things in the Aux bank as unshifted keys: number keys; syntax keys such as enter, begin, end, and cancel; the shift key; and the most common function keys. We placed the less frequently used function keys in the shifted version of the keys, and rarely used functions in the double-shift slots. So what if the Bitwise Exclusive Or is three keystrokes away?

General users won't care because they never use it, and hackers will appreciate that it's there at all!



One lesson we learned the hard way: make sure your cancel or clear key works in all shift states and that it resets the shift state. Otherwise, the user doesn't have a foolproof way to get the keyboard back in a known state.

We also found out two things about people versus keyboards:

1. People are always terrified that they are going to press the wrong button and wipe everything out.
2. People are always pressing the wrong buttons and wiping everything out!

(At least one thing about user interfaces is consistent!) We combatted this problem in three ways:

1. We made the keys that modified the setup, shifted keys. That way, it would be more difficult to accidentally modify the setup.
2. We added a way to lock all functions that modify the setup. Once locked, the user could not modify the setup without first entering a password.
3. We removed any possibility, even with a password, that a keyboard key press could invoke the “memory clear” function. We finally decided to make this self-destruct function require the user to insert a special plug in the rear of the unit. We thought about doing a “hold this key down for ten seconds while powering on the unit” kind of thing. But, we envisioned someone sitting on the remote control during a power outage, the power comes back on and....

Heed this advice: make it difficult to change setup parameters and nearly impossible to clear or reset all the parameters at once. Your users will appreciate it (in the long run) and your technical support people will appreciate it at all times.

While the maxi-mega-remote is fine for the power-craving technoliterate and average five-year-old, a large portion of the population remains keyboard challenged. For these folks, we provide a ten-button remote. No banks, no shifts, and you can't write programs on it. But you can (from the other remote) set up each key to run any program or infrared code.

CALCULATING KEY COUNT

Here's how you can calculate the number of keys you need. If you have

multiple banks, do the calculation separately on each bank. The descriptions and counts in Table 1 represent our specific implementation.

Given the above numbers, you can then determine how many shifted keys you have to work with. Simply add lines 1, 2, 3, and 7.

$$\text{Total shift functions available} = L1 + L2 + L3 + L7$$

In our case, there are 29 total shift functions.

To get the total number of functions available, multiply this number by the number of shift states you will support (1 for single shift, 2 for a double shift, etc.) and add line 1.

$$\text{Total functions available} = ((L1 + L2 + L3 + L7) * \text{Shift states}) + L1$$

In our case, using only a single shift would allow 43 functions, which is not enough. A double shift, however, provides 72 functions.

COMMAND FORMAT

You may be asking yourself what syntax keys are and why you need them. If you're really astute, you're also pondering the relative merits of prefix-versus-postfix notation.

Suppose I want to turn on light number five. (I could assign this function to one of the programmable keys, but for this discussion, assume we're doing things "manually.") In prefix notation, I press:

5 On

Since the arguments are entered before the command, the system can execute the command when I press On. If I also want light seven on, I press:

5 Enter 7 On

However, if I use a postfix notation, I press:

On 5 Enter 7 End

As you can see, the postfix notation involves an extra keystroke.

Which notation is better? I performed an informal survey of both people and products and discovered the following:

- Average people are pretty much evenly split on their preference. Programmer types (being particularly *unaverage* when it comes to evaluating technology) tend to prefer postfix.
- Products are also evenly split. (For every prefix user interface, I found a postfix version.) Amazingly, some user interfaces don't seem to be one way or the other. They are a hodgepodge of random syntaxes!

I also learned that everyone had a preference, one way or the other, and upon such strong feelings have truly nasty wars been fought. I urge you to get this issue behind you as quickly as possible.

I don't want to be accused of starting any religious wars here, but let me throw in my two cents' worth. After a very lively debate, my design group decided that prefix notation was more appropriate for what we

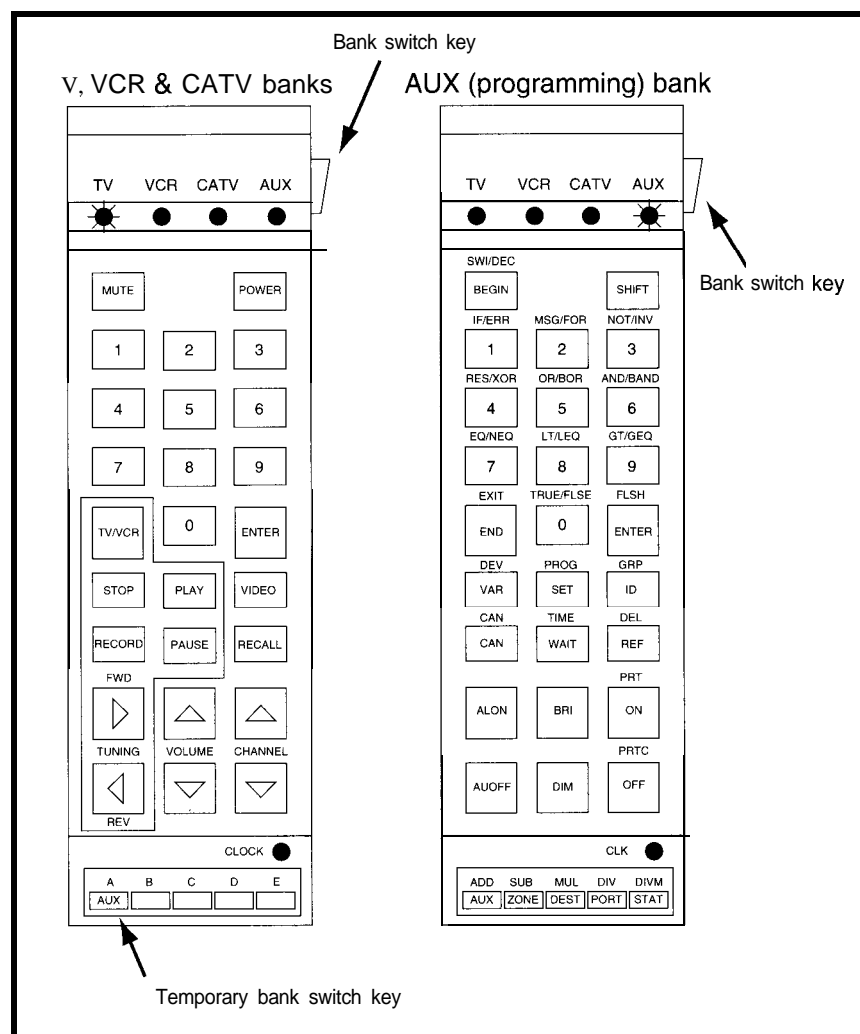


Figure 4: By switching to the Aux bank, the 64 functions and numerals needed for programming a whole house are available.



call immediate, one-function commands. Since the most frequently used commands involve one keystroke for the argument and one for the command, saving an extra End keystroke is significant.

But, since most programmers prefer postfix notation, we opted to use postfix for programming. (Actually, we weren't given much choice since CAL is a postfix-based language.) But, we discovered an elegant way to allow prefix notation for single commands! In its quest for terseness, CAL introduces the concept of a "default command." If arguments are encountered without a preceding command, the default command is executed on those arguments.

Our elegant contribution involves two steps:

1. We defined the default command as "current group." This command simply stores its arguments for future reference.
2. We provided that many commands, when executed with no arguments, use the stored current group as their arguments.

The user can now type arguments, followed by a command, and the command executes

on those arguments—exactly like prefix notation. The programming structure, however, remains universally postfix.

Whether the users appreciate this design construct remains to be seen. By the way, CAL really works quite well as both a user interface and a programming language! It's sometimes hard to believe it came out of a committee.

SUMMARY

Designing the layout and syntax of a complex key board user interface is ultimately all about priorities. You have to decide which features are most important and recognize that less-important features may not make it into the final design. Similarly, you have to ask which functions are most important. Less-important functions can be relegated to obscure multikey-stroke positions.

There is a definite sequence to laying out keys on a keyboard. Failure to follow this sequence can result in a hopeless mess.

First, you must have a firm understanding of the command syntax you wish to implement. (Writing out many examples seems to be the

best way to test and describe command syntax.) Next, you need to define any bank switch, temporary bank switch, and shift keys you will be using. Go ahead and assign the key if you can since these will be among the most used keys. Then, you can assign any syntax keys such as the Enter and Cancel key. Only when all this is done can you assign the other keys.

I got bogged down because I hadn't applied any priorities and because I didn't have this design sequence spelled out. Now that you know all the pitfalls, it's your turn.

Jeff Fisher is president of HomeTech Solutions, a home automation manufacturer and retailer in San Jose, California. He may be reached at (408) 257-4406 or 71431.3343@compuserve.com.

IRS

425 Very Useful

426 Moderately Useful

427 Not Useful

Find out how you can add intelligence to any home, at a cost that's within your budget.

LIVING WITH AN INTELLIGENT HOME
will change the way you live

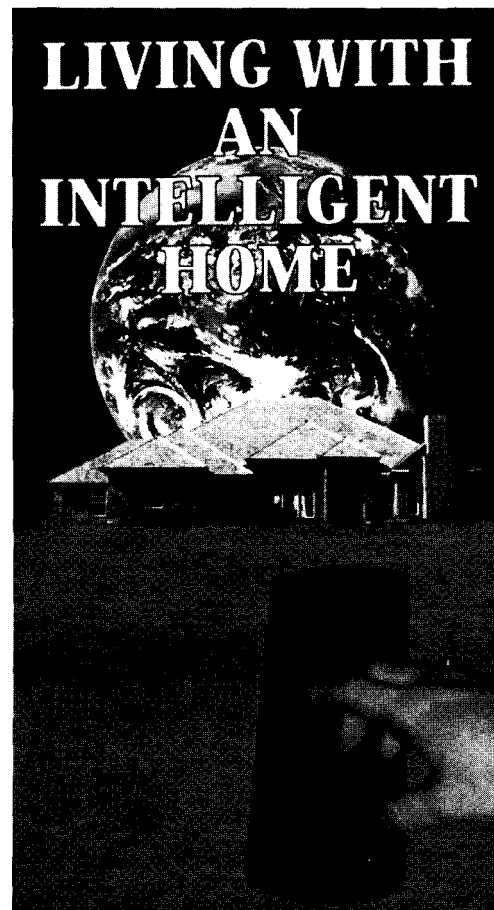
*Written by David Gaddis, author of **Understanding & Installing Home Systems***

This VHS video cassette retails for \$24.95 plus \$5 s&h.

It is being offered to Circuit Cellar INK subscribers for only **\$17.95** plus \$4 s&h (U.S.)

ORDER TODAY!

Don't let this exciting technology opportunity pass you by!



Take a tour through modern technology in today's home...

Circuit Cellar, Inc.

4 Park St. • Vernon, CT 06066

Tel: (203) 875-2751

Fax: (203) 872-2204

Y

ou know the fire drill. You've just started a movie on the VCR and the phone rings. Should I or should I not answer it, pause the tape or let it roll

and hope to dispose of the call quickly? Is it an important call from a friend or relative or just another annoying call from MegaBank offering me a preapproved credit card...again?

Even if you have a telephone answering machine (and these days, who doesn't?), you still end up being distracted by the ringing phone. After one hundred years of living with the telephone, it's astounding to think how little the basic operation has changed—the phone rings, we answer it. It's the human equivalent of a nonmaskable interrupt (go on, just try to ignore that ringing phone).

So, as you restart the VCR after telling the MegaBank representative that you don't want another credit card, thank you, your mind wanders from the movie as you ponder this problem of the telephone. You think, "If I had a gadget that could screen my calls, like a voice-mail system, I wouldn't be interrupted unless the call was urgent. The gadget could shunt off solicitors—"Telemarketers press 9 now"—and have passwords for friends so they could get through directly. It could be run with a PC."

Of course it sounds great. As the movie regains your attention, you only wish that someone would make one of these.

DSP TO THE RESCUE

While flipping through the pages of a Dallas Semiconductor data book, we came across a chip that could make such a system possible. The DS2132A Digital Answering Machine Processor from Dallas Semiconductor is a special-purpose DSP with a very useful mix of features. It is capable of voice compression and decompression, DTMF detection and generation, call progress tone generation, and if that weren't enough, it can also make music (on a limited scale, anyway).

Computer, Get That Phone

A PC-based Voice-telephone Interface

ROBERT M. LUZENSKI & JACK IVEY

Robert and Jack dream of viewing a movie free of telemarketing interruptions. With the new **DS2132A** Digital Answering Machine Processor, the realization of such a dream is much closer.

Of course, you pay for all this capability with complexity of operation. The chip has two main interfaces: a PCM port that connects to a telephone CODEC to transfer digitized voice data at the standard rate of 64 kbps (8 bits per sample \times 8k samples per second) and a Compressed Data (CD) port that exchanges compressed voice data, commands, and operating status with a microprocessor.

This article offers an example of the 2132A's basic hardware and software design. You can use this information as the basis of more complex designs including the "Telephone Wonder Gadget" presented in the introduction.

But, let's start with the hardware description before we get interrupted by the phone again.

HARDWARE

The hardware is divided into two sections—one analog and the other digital. The 2132A/CODEC combination sits between the two sections with analog voice on one side and digitally compressed voice on the other. The analog section provides the telephone-line interface and auxiliary analog connections. The digital section provides the interface between the PC ISA bus and the 2132A.

Figure 1 covers the ISA bus interface including the combined CD port timing and interrupt-generation circuitry. Figure 2 presents a schematic of the analog section, 21.322, and PCM port-timing hardware.



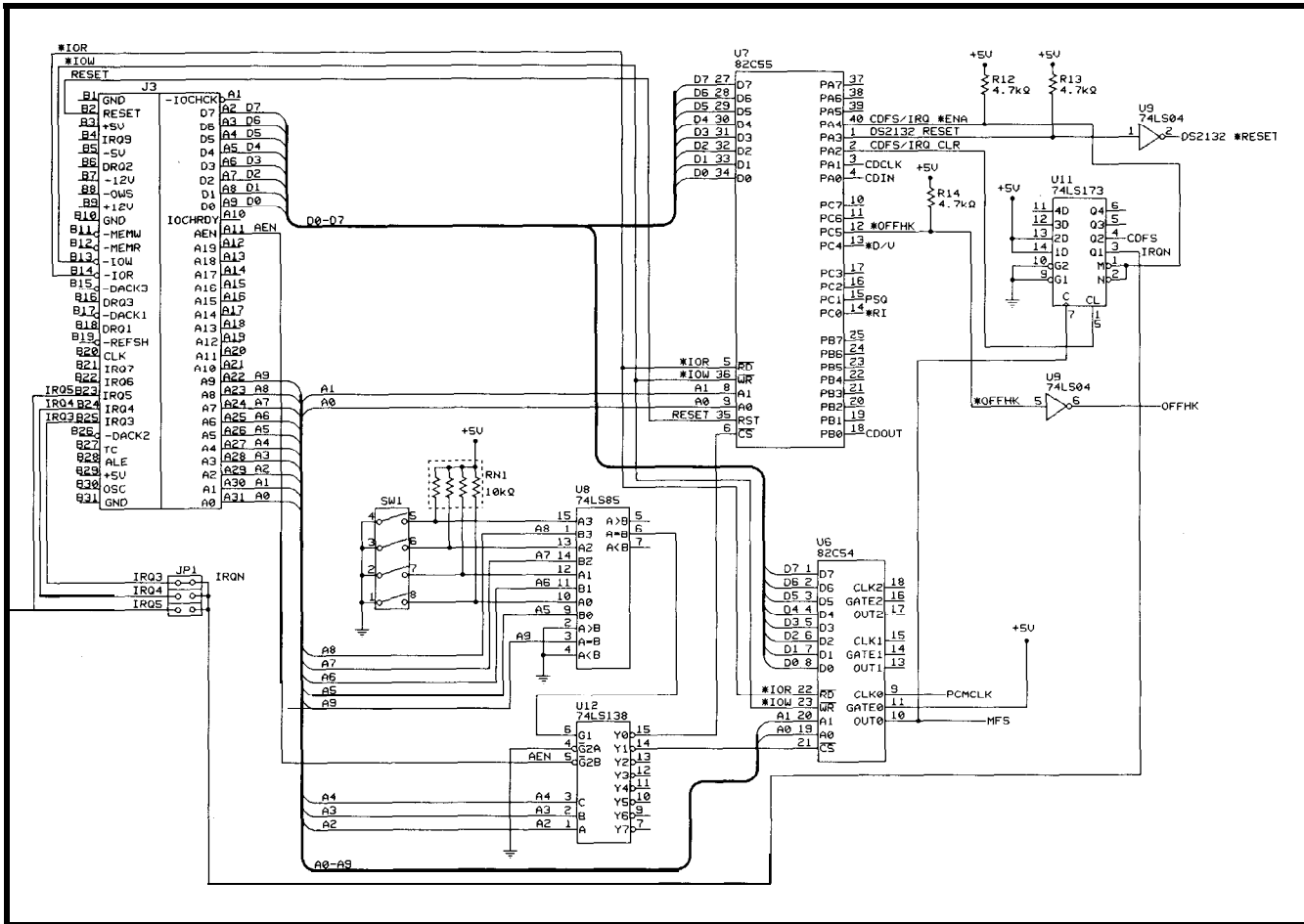


Figure 1: I/O ports on the 82C55 provide the main interface between the PC bus and the DS2132A. The 74LS173 generates the IRQ and CDFS signals on the rising edge of the Master Frame Sync, and allows them both to be cleared under software control.

ISA BUS INTERFACE

The PC ISA bus interface provides parallel ports for communication with the 2132A and control of the DAA. In addition, it provides the necessary timing signals for the CD and PCM ports and an interrupt signal, which can be set to one of three IRQ lines on the bus. The parallel ports and timer are I/O mapped with a DIP switch to select the base address.

I/O decoding for the ISA bus must deal with the 10 least-significant address lines (A0-A9), the I/O read (IOR) and write (IOW) lines, and the address-enable line (AEN). A 74LS85 decodes the base address by comparing A5-A8 with the DIP-switch settings. A9 must be high for decoding to occur. The output from the address decoding enables a 74LS138, which decodes A2-A4. AEN must also be low to enable the 138. This prevents actuation of the chip-select lines when a DMA cycle occurs on the bus. The resulting chip selects enable the 82C55 parallel I/O chip and the 82C54 timer.

Parallel I/O using the 82655 is straightforward enough. The hardest part is looking up which of the 192 operating modes to select, and we've already done that for you. The three 8-bit ports on the chip (A, B, and C) are all byte-wise configurable as either inputs or outputs. Port C can also be split so that half the port pins are inputs and half outputs. In our design, port A controls the 2132A and is configured as an output port. A single bit of port B is used as an input from the 2132A, so port B is configured as an input port. Port C is dedicated to supporting the DAA and is operated in the split mode.

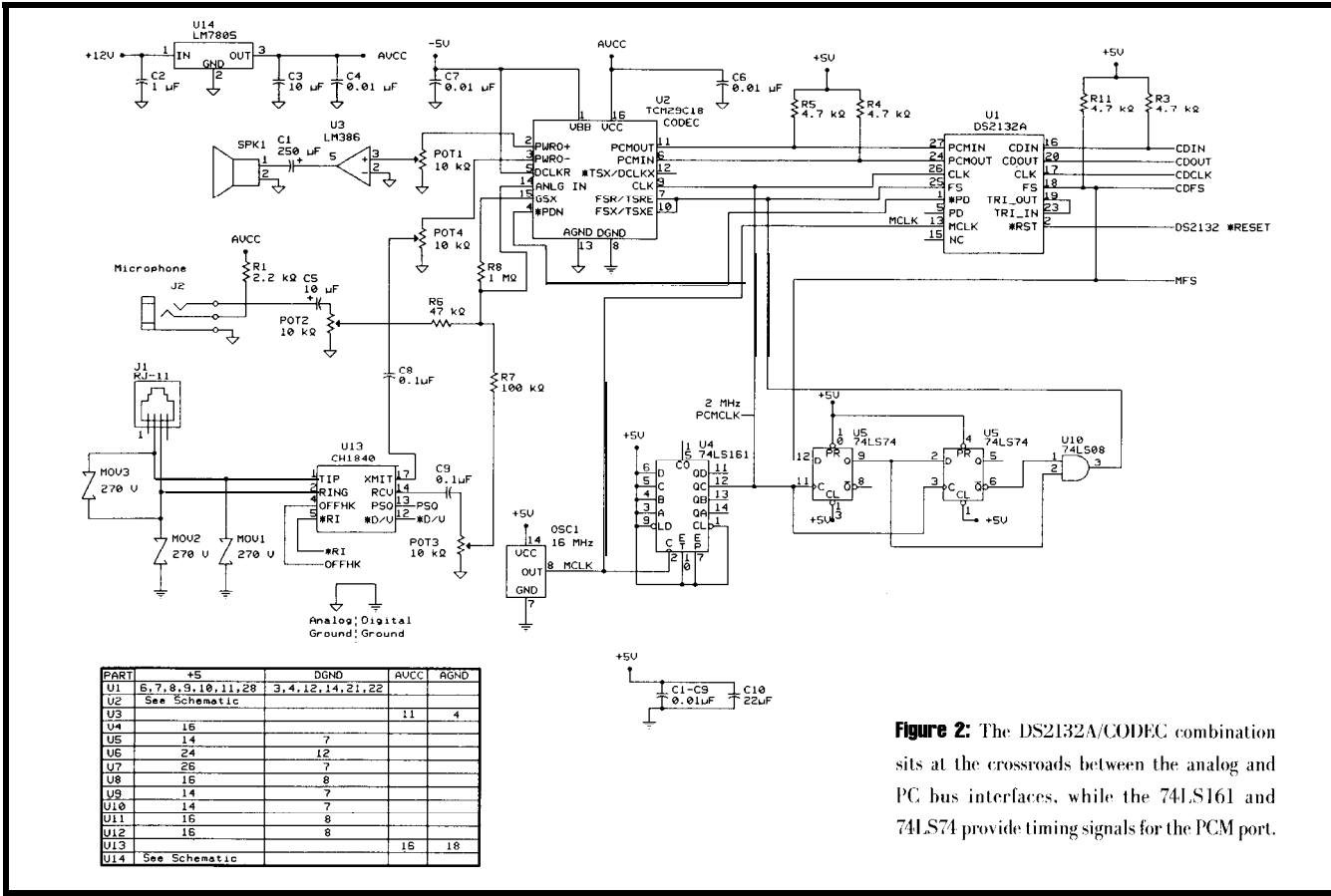
Well-behaved PC ISA bus boards do not make strange noises on powerup, take the phone line off-hook and scream nonsense into it, or generate interrupts without software loaded to handle them. To make our board well-behaved, we ensure that the 2132A is held in reset and the interrupt line is disabled when the system is first powered up.

TIMING IS EVERYTHING

Both the PCM and CD ports require frame sync and clock signals to control their operation. The signals are referred to as PCMFS and PCMCLK in the case of the PCM port, and CDFS and CDCLK for the CD port. The basic timing for the CD port can be seen in Figure 3.

consists of one cycle of CDCLK with CDFS transfer the command/status. CD port is a toggle only

There, that port is less complicated than the simultaneously, it without any effort on the part of the software. The data sheet indicates that the Frame Sync and CLOCK pins from both



PART	+5	DGND	AVCC	AGND
U1	6,7,8,9,10,11,28	3,4,12,14,21,22		
U2	See Schematic			
U3			11	4
U4	16	7		
U5	14	7		
U6	24	12		
U7	26	7		
U8	16	8		
U9	14	7		
U10	14	7		
U11	16	8		
U12	16	8		
U13			15	18
U14	See Schematic			

HCS II Home Control System

EXTENDED SYSTEM

Basic System

PL-Link Module

PS-485

HCS II BASIC SYSTEM

PS12-1 Power Supply

Buffer/Terminator I/O Board

TW523

Energy Management

Security and Alarm

Coordinated Home Theater

Coordinated Lighting

Monitoring and Data Collection

Get all these capabilities and more with the Circuit Cellar HCS II. Call, write, or FAX us for a brochure. Available assembled or as a kit.

CIRCUIT CELLAR, INC.
 4 Park Street, Suite 12 • Vernon, CT 06066
 Tel: (203) 875-2751 • Fax: (203) 872-2204

Circuit Cellar HCS

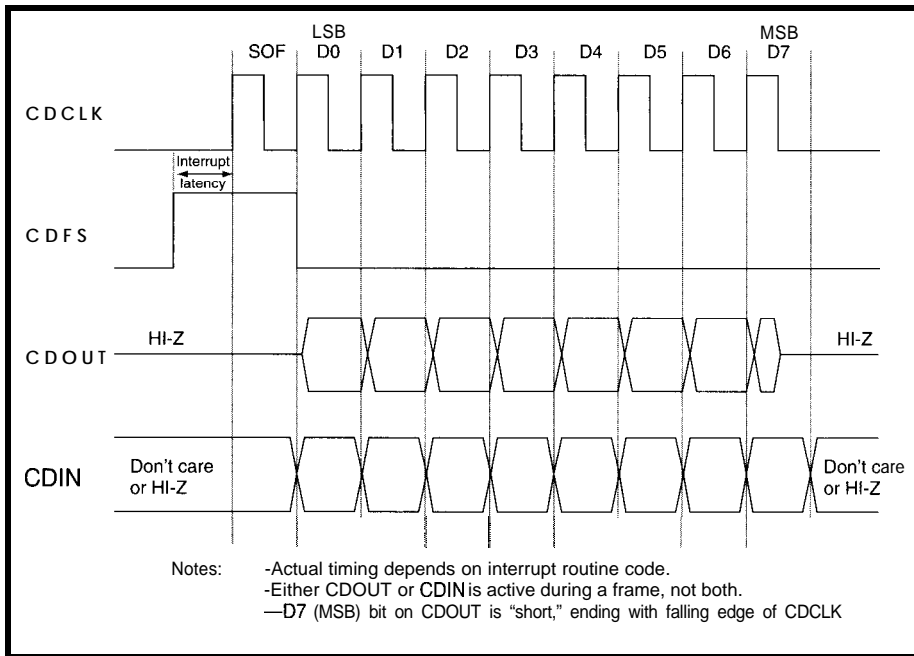


Figure 3: The Compressed Data (CD) port transfers data between the DS2132A and the PC ISA bus.

ports should be tied together. However, in actual practice the signals do not have to be tied together as long as a few simple requirements are met. These requirements are that the PCMCLK frequency must be greater than the CDCLK frequency, and that the PCMFS anti-CDFS must occur within 2.0 μ s of each other. An application note available from Dallas Semiconductor describes the Frame Sync and Clock requirements of both ports in greater detail than the data sheet, and also shows how to connect several CODECs from various manufacturers.

For our board, the CDCLK frequency is determined by ISA bus timing and low-level software, and the frame-sync requirement is guaranteed by the hardware design. The software sets up the 82C54 timer to generate an 8-kHz Master Frame Sync (MFS) signal from the 16.0-MHz Master Clock (MCLK) driving the 2132.4. The MFS signal then drives separate circuits to trigger both CDFS and PCMFS.

Figure 5 is a block diagram that illustrates the generation and distribution of these timing signals. The 2.0-

MHz PCMCLK is generated by the 74LS161 set up to divide MCLK by eight. The PCMFS signal is generated with a 74LS74 that has PCMCLK and CDFS as inputs, as suggested in the Dallas Semiconductor application note on CODEC interfacing. The design ensures timing requirements for the PCMFS are satisfied.

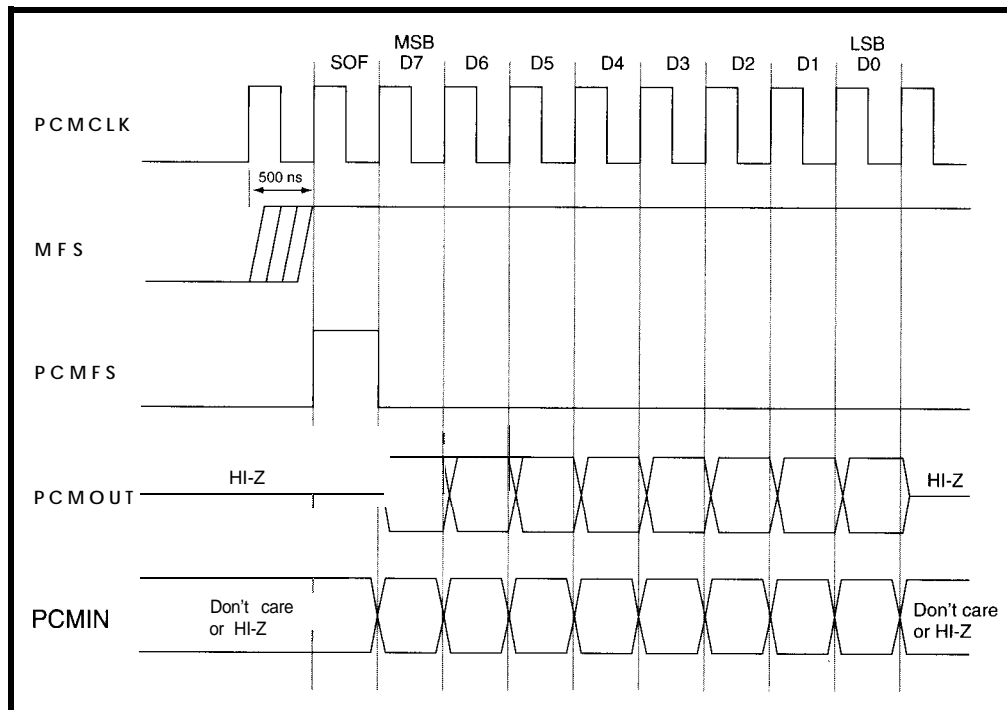


Figure 4: Unlike the CD port, the PCM port transfers data in both directions at the same time with the most significant bit first.

We use a 74LS173 to generate both the CDFS and IRQ signals. The reasons for this arrangement require some explanation. With IRQs, good ISA bus design demands that an IRQ line only be asserted when the device is actually being used. This allows more than one device to be connected to the same IRQ as long as only one is enabled at a time.

The '173 is a quad latch with tristate outputs, chosen to allow the software to release the IRQ line when the board is not being used. The CDFS/IRQ *ENA line from port A of the 82C55 connects to the tristate control of the '173, and serves as an enable for both CDFS and IRQ.

When this enable line is true, CDFS is active and IRQ is asserted on the ISA bus IRQ line selected by JP1. If false, CDFS is pulled high by a resistor, and IRQ is left floating. In operation, both CDFS and IRQ are triggered by MFS. When the interrupt routine executes, it controls the exchange of data with the CD port by driving CDCLK and either writing to the CDOUT line or reading from the CDIN line. The routine also toggles CDFS/IRQ CLR to reset CDFS and IRQ during the transfer.

Figure 5: The 16.0-MHz MCLK is divided using an 82C54 timer and some LSTTL to generate the timing signals required by the PCM and CD ports.

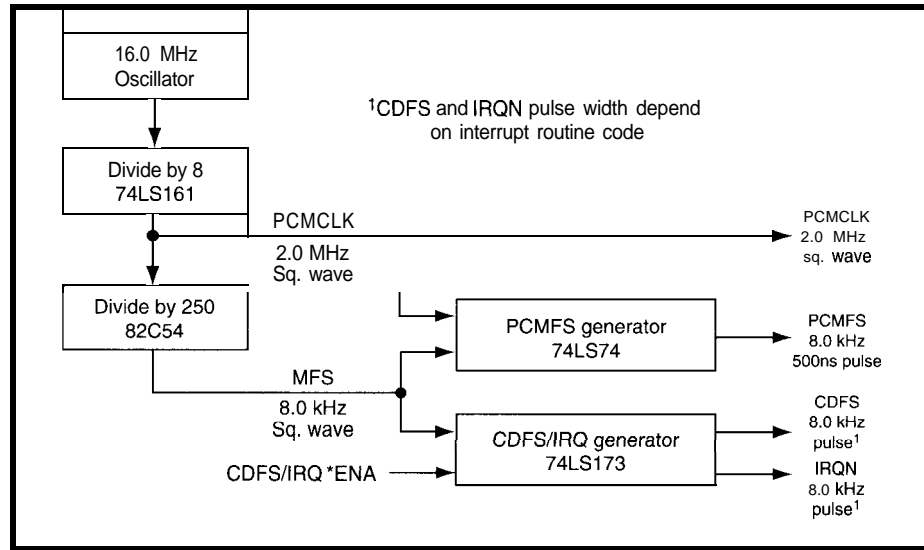
ANALOG IS EASY (NOT!)

Actually, it always looks easy after you get it working right.

Our design has two audio inputs and two audio outputs, with the inputs coming from a microphone or the phone line (through the DAA), and the outputs going to a speaker or the phone line (again through the DAA).

At first glance, it seems that some switching is required for the inputs. However, since the microphone input is intended to record prompts and messages when the phone is not in use, only one of the inputs is expected to be active at a time. This allows the two inputs to be summed instead of switched, saving some hardware and software.

The CODEC has two output pins, which can be used as either a single



differential output or as two separate ground-referenced outputs. We used the latter method to provide individual outputs for both the DAA and speaker. The CODEC outputs purports to be a power output, but it can only drive a 300-ohm load. While this is fine for the IN-1 XMIT input, it is not so good for the speaker. As a result, we added the LM386




audio amplifier set for a gain of 20, which nicely drives an 8-ohm speaker.

The input to the CODEC is an uncommitted op-amp with pins for the output and inverting input, and the noninverting input tied to ground. This allows the gain to be controlled with the input and feedback resistors. It also enables more than one signal to be mixed using a summing junction.


The only 8051/52 BASIC compiler that is 100% BASIC 52 Compatible and has full floating point, integer, byte & bit variables.

- Memory mapped variables
- In-line assembly language option
- Compile time switch to select 8051/8031 or 8052/8032 CPUs
- Compatible with any RAM or ROM memory mapping
- Runs up to 50 times faster than the MCS BASIC-52 interpreter.
- Includes Binary Technology's SXA51 cross-assembler & hex file manip. util.
- Extensive documentation
- Tutorial included
- Runs on IBM-PC/XT or compatible
- Compatible with all 8051 variants
- **BXC51 \$ 295.**

508-369-9556
FAX 508-369-9549



Binary Technology, Inc.
P.O. Box 541 • Carlisle, MA 01741



Think Fast.

COSMIC C Tools Speed Development

Fast, efficient, reliable code generation has distinguished COSMIC C cross compilers for over a decade. Now, source level debugging is easier than ever with ZAP, its intuitive interface is uniform across all targets. Integrated under the Windows Development Environment, COSMIC C tools deliver the fastest, most reliable performance you can get. For technical details, email an evaluation package, call **617-932-2556**. In Europe +33 (1) 43,99,53,90

- Features include:
- All-in-one: our Speedy Development environment
 - ZAP
 - Fully portable
 - Regularly updated by Source
 - In-line Assembly
 - Full-featured, full-featured
 - Integrated Development Environment
 - X-Target: Visual C++
 - Compiler: DVA, M-Assembler, BDM, Turbo C++
 - Supports all Windows, NT, HP-OS/2, Motif

68000, 68XXX, 68110, 68110 (L, 68009, Z80, Z180, 64180, 8051, NEC K0/K2



Source and 100% compatible with Windows, Macintosh, and OS/2. Copyright © 1995 COSMIC Software, Inc. All rights reserved. Printed in the USA.

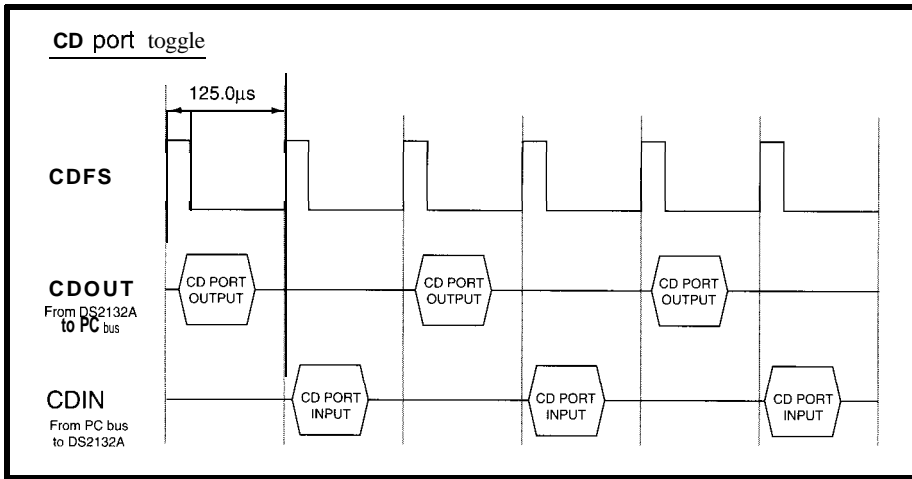


Figure 6: The CD port toggles between input and output on each successive frame

We took advantage of this capability to sum the microphone and DAA signals. Audio from either source is amplified and digitized by the CODEC. The input resistor network is set up so the microphone has a maximum gain of 20 and the DAA a maximum gain of 10. Additionally, pots on the input and output lines allow fine tuning of the various gains, and if needed, microphone power is supplied through the middle terminal of the stereo plug.

THE HARDWAY IS OUR HOBBY

Sonic basic precautions are required when combining digital and analog circuits in a single system, especially if it will be inside a PC. An amazing variety of amusing warbles, whistles, and bagpipe sounds find their way into analog circuitry if you're not careful. Those of us who spend most of our time in the digital world (or worse, software) always have to find this out the hard way.

Providing clean power and ground goes a long way toward eliminating most of the problems. To that end, we ignore the +5-V regulated power coming from the PC bus. Instead, we power all the analog circuitry from an onboard +5-V regulator attached to the +12-V PC bus supply. This eliminates the 50-mV or more of ripple and hash caused by the switching transients from ICs, hard drives, and the like.

Additionally, we provide separate analog and digital return (ground) paths, which helps prevent digital-switching transients from creating a noise voltage on relatively small analog signals such as the microphone input. Analog and digital grounds are connected at only one point. We sprinkled 0.01-µF by pass capacitors around the digi-

tal ICs, keeping them close to the chips, and making sure that the larger chips and clock oscillator got their very own.

Finally, you may notice that the MOV (metal oxide varistor) connections appear to violate the separation of grounds principle. This is because the MOVs need to be connected to a chassis ground point, such as the board's metal bracket. The good news is that this minor violation doesn't seem to add any noise to the circuit.

DS2132A OPERATION

The software accompanying this article provides a low-level interface to the hardware. This interface is intended to serve as

a foundation for PC-controlled voice applications. Most details involved in communicating with the 2132A have been handled, leaving the application software with the less-demanding task of operating on a functional level. We'll describe the routines in the low-level interface shortly, but first let's look at how to operate this beast.

Internally, the 2132A has two data paths, one for recording and the other for playback. The record path takes in digitized data from the PCM port and compresses it for output on the CD port. The playback path takes in compressed data from the CD port and uncompresses it for output on the PCM port.

Roth paths have a gain block that can be controlled with commands to the 2132A, and only one of the paths can operate at a time. A loop-back mode is available, which takes digitized data from the record path and inserts it in the playback path, replacing anything that would otherwise be coming from the playback channel. The loop-back path includes both the record and playback gain blocks, and has no effect on the record channel.

CD PORT TWO-STEP

Compared to the PCM port, the CD port is fairly complicated. The

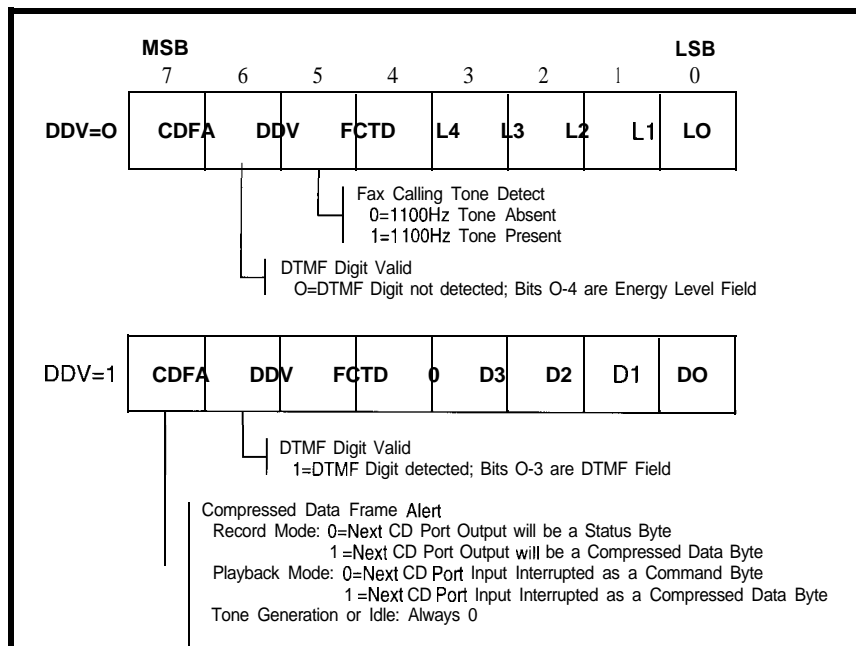


Figure 7: Interpretation of the CDFA bit depends on the current operating mode. The meaning of bits 0-4 are determined by the value of the DDV bit.

Table 1: A summary of the command bytes supported by the DS2132A.

PCM port merely transfers digitized data back anti forth, but the CD port mixes command, status, and digitized data bytes on the same port. In addition, unlike the PCM port, which works in both directions at the same time, the CD port toggles between input and output with each successive pulse of CDFS. This significantly complicates operation of the port, requiring the software to go to some trouble to keep it all sorted out. Figure 6 illustrates the basic operation of the CD port.

There are three types of data bytes communicated over the CD port. Command bytes are inputs to the port and tell the 2132A what

to do. Status bytes are outputs and provide status information from the 2132A. Finally, compressed data bytes can go in either direction and contain the actual compressed voice data.

COMMAND BYTES

While there are over 100 individual command bytes, they can be classified into four basic groupings. Operational commands control the basic modes of the 2132A. Tone-generation commands direct the 2132A to output any of a selection of tones through the playback path. The chip includes all sixteen **DTMF** tones, several call-progress tones, and a selection of musical notes. Record commands control the gain and silence threshold of the record path and select the compression rate and **silence** compression options. Playback commands control the **gain** of the

playback path, designate the compression rate of the data, and select the DTMF echo-cancellation option. Table 1 provides a summary of available command bytes.

STATUS BYTE

The format of the status byte is illustrated in Figure 7. The status byte actually performs two separate functions, depending on the value of the DTMF Digit Valid bit (DDV, bit 6). Normally, the DDV bit is set to 0 and the lower 5 bits of the status byte provide an indication of the energy level of the incoming voice **signal**. The interpretation of the energy-level field (L0-L4) is shown in Table 2.

However, if a valid DTMF tone is received, the DDV becomes 1 and the lower 4 bits of the status byte are interpreted as the DTMF field (D0-D3) with bit 4 set to 0. The Fax Calling Tone Detect bit (FCTD, bit 5) indicates when a fax-calling tone is received. **Fax**



a) Operational Command Bytes			
Mode Control Commands		Special Mode Commands	
00	No Update	08	Enter Loopback Mode
FF	No Update	09	Exit Loopback Mode
BE	Idle	04	Enter Power-Down Mode
		05	Exit Power-Down Mode
b) Record Command Bytes			
Record Commands		Record Gain Settings	
21	4:1 Compression Rate (16 kbps)	4A-40	+30 dB through 0 dB (in 3 dB steps)
23	8:1 Compression Rate (8 kbps)	5F-56	-3 dB through -30 dB (in 3 dB steps)
25	4:1 Compression Rate (Silence Compression)	Silence Threshold Settings	
27	8:1 Compression Rate (Silence Compression)	10-1 F	-50 dBm through -11 dBm (step size varies)
c) Playback Command Bytes			
Playback Commands		Playback Gain Settings	
20	4:1 Compression Rate (DTMF Echo Cancellation Off)	6A-60	+30 dB through 0 dB (in 3 dB steps)
22	8:1 Compression Rate (DTMF Echo Cancellation Off)	7F-76	-3 dB through -30 dB (in 3 dB steps)
28	4:1 Compression Rate (DTMF Echo Cancellation On)		
2A	8:1 Compression Rate (DTMF Echo Cancellation On)		
d) Tone Generation Command Bytes			
DTMF Tones		Call Progress Tones	
80	DTMF 0 (941+1336 Hz)	90	Dial Tone (350+440 Hz)
81	DTMF 1 (697+1209 Hz)	91	Ringing Tone (480+440 Hz)
82	DTMF 2 (697+1336 Hz)	92	Busy Tone (480+620 Hz)
83	DTMF 3 (697+1477 Hz)		
84	DTMF 4 (770+1209 Hz)	Musical Tones	
85	DTMF 5 (770+1336 Hz)	B4-BA	Musical Note A (440 Hz) through G (784 Hz)
86	DTMF 6 (770+1477 Hz)	BB	Musical Note A one octave higher (880 Hz)
87	DTMF 7 (852+1209 Hz)	BC	Musical Note B one octave higher (988 Hz)
88	DTMF 8 (852+1336 Hz)	94-9A	Bright Musical Note A (440+1320 Hz) through G (784+2352 Hz)
89	DTMF 9 (852+1477 Hz)	9B	Bright Musical Note A one octave higher (880+2640 Hz)
8A	DTMF A (697+1633 Hz)	9C	Bright Musical Note B one octave higher (988+2974 Hz)
8B	DTMF B (770+1633 Hz)		
8C	DTMF C (852+1633 Hz)	Other Tones	
8D	DTMF D (941+1633 Hz)	93	400-Hz Tone
8E	DTMF * (941+1209 Hz)	9E	1004-Hz Tone
8F	DTMF # (941+1477 Hz)	9D	1400-Hz Tone

machines adhering to ITU-T Recommendation T.30 transmit an 1100-Hz tone for 0.5 seconds to identify an incoming nonvoice call to the receiver. The software can use this information to handle a fax call appropriately.

Finally, the Compressed Data Frame Alert bit (CDEA, bit 7) indicates that a compressed data byte is needed or will follow. This alert is used during recording and playback. It is described in greater detail in the sections covering those modes.

COMMAND BASICS

Using command bytes with the 2132A is relatively simple. With the exception of the No-Update command (00H/FFH), all commands need to be sent only once to become effective. The record, playback, and tone-generation commands continue to operate until another such command or the Idle command (BEH) is sent. The various setting commands (record/playback gain,

					Received
L4	L3	L2	L1	LO	Energy Level
0	0	0	0	0	<-48 dBm ₀
0	0	0	1	0	-45 dBm ₀
0	0	1	0	0	-42 dBm ₀
0	0	1	0	1	-39 dBm ₀
0	0	1	1	0	-36 dBm ₀
0	0	1	1	1	-33 dBm ₀
0	1	0	0	0	-30 dBm ₀
0	1	0	0	1	-27 dBm ₀
0	1	0	1	0	-24 dBm ₀
0	1	0	1	1	-21 dBm ₀
0	1	1	0	0	-18 dBm ₀
0	1	1	0	1	-15 dBm ₀
0	1	1	1	0	-12 dBm ₀
0	1	1	1	1	-9 dBm ₀
1	0	0	0	0	-6 dBm ₀
1	0	0	0	1	-3 dBm ₀
1	0	0	1	0	0 dBm ₀
1	0	0	1	1	+3 dBm ₀
1	0	1	0	0	+6 dBm ₀
1	0	1	0	1	+9 dBm ₀

Table 2: Bits 0-4 of the status byte are interpreted as the received energy-level field when the DDV bit is 0.

silence threshold) are effective until changed by another such setting command. The No-Update command is the default command and should be sent when no other command is needed.

TONE GENERATION

The easiest type of commands to use are the tone-generation commands. Figure 8 shows the exchange of command and status bytes needed to generate tones on the playback channel. The 2132A starts generating a tone when it receives the tone command. No-Update commands can be used to control the length, and an Idle command ends the tone. That's all there is to it.

RECORD MODE

In record mode, the 2132A compresses digitized voice data from the PCM port and outputs it to the CD port. The exchange of commands and data on the CD port are shown in Figure 9. The CDFA bit in the status byte indicates when a compressed data byte is ready.

When CDFA is 1, the next output frame from the 2132A is a compressed data byte rather than a status byte. Once recording starts using one of the four record command bytes, 10-1 update commands can be sent to continue recording as long as needed. The gain can be controlled during recording by monitoring the energy-level field in the status byte

and adjusting the gain accordingly using the record-gain-setting commands. Recording stops by sending an Idle command.

COMPRESSION RATES

The 2132A can compress 64 kbps of voice data at fixed ratios of either 4:1 or 8:1, resulting in data rates of 16 or 8 kbps. These data rates can be reduced even further by applying the silence-compression option available with record commands 25H and 27H.

Silence compression works by replacing signals which fall below a silence threshold with a code in the compressed data stream representing the period of silence. This results in a lower data rate because the silence code is shorter than the digitized data it replaces. No special effort is required for playback of data recorded with silence compression. The same playback command controls normal anti-silence-compressed data.

We will not go into detail about how to set the threshold value since it is transparent to the low-level software interface presented in this article. In fact, a careful read of the data sheet does not supply much enlightenment on the topic of silence compression either.

The only hint of information is contained in the "Command Byte Options" table which lists all the command bytes available. There you find reference to recording at various rates with curious names such as premium, intermediate, standard, and extended. Standard and extended are the recording modes that use silence compression. The tone commands come attached to the numbers 9.8 kbps and 4.9 kbps, and therein

lies a story.

The key to using silence compression to reduce data rates is getting the proper setting of the silence threshold. For best recording quality, the threshold should be set actively, based on the energy level of the recorded signal as indicated in the status byte. An algorithm to accomplish this task must balance a multidimensional problem involving data rate, sound quality, and algorithm complexity to arrive at a useful setting for the silence threshold.

An application note from Dallas Semiconductor describing the silence-compression option characterizes normal speech as having 20-40% silence, depending on the speaker. The 9.8- and 4.9-kbps numbers in the data sheet result from silence compression of just a notch under 40%. Since the actual data rate depends on the setting of the threshold and the content of the signal itself, the unqualified claim to the most favorable compression rates is a bit misleading.

That said, the application note on silence compression does contain a detailed analysis of the threshold problem, including a solution with C source code. We leave it to the more adventurous to experiment with silence compression and to see how well they strike the balance between data rate, sound quality, and CPU cycles.

PLAYBACK MODE

For playback mode, the software sends the compressed data to the CD

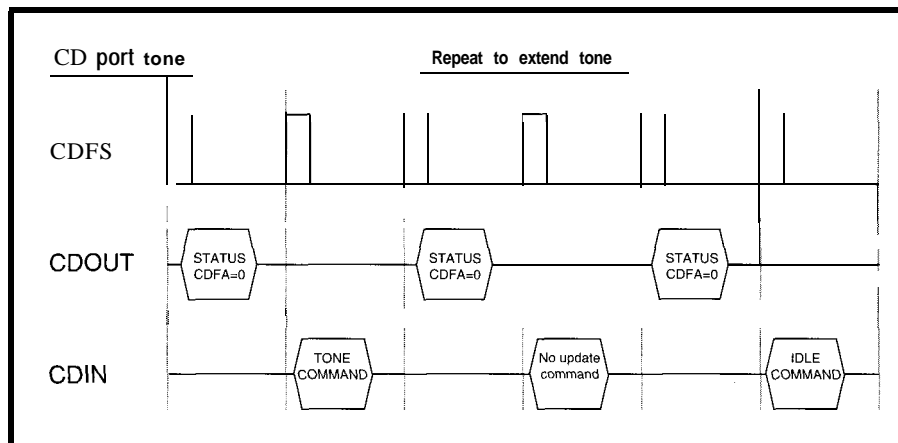


Figure 8: Tones are generated by sending a tone command followed by No-Update commands to extend the tone to the desired length.

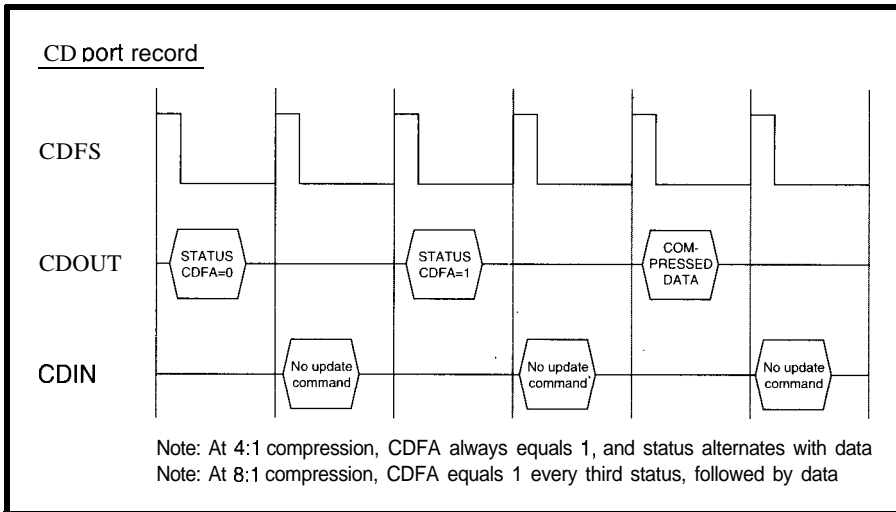


Figure 9: When recording, a status byte with CDFA set to 1 indicates that the next CD port output frame contains a compressed data byte rather than a status byte.

DTMF ECHO CANCELLATION

The frequency content of normal speech is capable of setting off the DTMF detector in the 2132A. You can easily see this by monitoring the status byte while recording your own voice. The DDV bit goes true every so often, indicating that a DTMF tone was detected in your speech. If the voice energy reflected back to the input by the phone line happens to contain the proper frequencies, it triggers a false indication of a DTMF digit in the status byte.

The DTMF echo cancellation feature avoids this by muting the playback when a DTMF tone is detected. If the DTMF tone goes away when the playback is muted, then the playback is the cause of the DTMF tone detection. The playback continues, although it will be muted until the DTMF tone passes. On the other hand, if the DTMF tone per-

port. The 2132A uncompresses the data and sends it along to the PCM port. Figure 10 shows the operation of the CD port in playback mode. In this case, the CDFA bit indicates when the 2132A needs another compressed data byte. The compressed data byte must be sent in place of a command byte in the frame immediately following the status byte.

There are four commands for playback, two at each of the available compression ratios. The choice involves DTMF echo cancellation. Strangely, this is another area where the data sheet is silent, but an application note fills in the blanks left by the data sheet. Notably, we find that echo cancellation is very important if you want to use voice prompts requiring DTMF responses.



Tech·Arts TEL 315·455·1003
 FAX 315·455·5838
 BBS 315·455·8728
 308 E. Molloy Road Syracuse, NY 13211 FOD 315·455·8368

- *Text to Speech Board w/ true serial I/O \$89
- *Temperature boards: w/16 temp. sensors \$239
 w/8 temp. sensors plus 8 analog inputs \$179
 both brds: -40°F to 146°F, w/true serial I/O
- *Digital I/O ISA cards: 46 I/O ports \$125
 96 I/O ports \$169
 192 I/O ports \$249
- Infrared transmitter & receiver pair \$89
 connect to PC w/digital I/O, 16 commands
- CPort ISA Serial Board w/ true 16550's \$129
 com1-coma & irq's 2,3,4,5,10,11,12,15
- 4-Servo controller board w/ true serial I/O \$89
- *Programmers X-10 Library for C \$95
 100+ functions, w/cable(PC to TW523)
- *Picture-In-Picture box w/TV Tuner \$159
- *Electronic Drapery Control use w/ X-10 \$139
- *Keyboard Multiplexer: 2 KBD's into 1 PC \$49
 chainable to allow multiple KBD's on 1 PC!
- *Fax-On-Demand w/Voice Mail: SW & HW \$249
 requires 286 or above w/harddisk and DOS
- *Windows NT TelecomFAX Personal SW \$129
- Windows NT TelecomFAX Server SW \$495
- Windows NT Modem Pooling Software \$795

Many more computer items available!
 Prices are subject to change without notice.

800·455·9853
 Visa • MC • Amex • COD

Call for a
FREE CATALOG!

Where Home Automation meets Computer Technology!

HOME AUTOMATION IS HERE TO STAY!



Get Your Copy of The Best Source of Home Automation Products Absolutely Free.

Largest Selection of Home Automation products in the World

Call 24hrs for Free 64 page Color Catalog

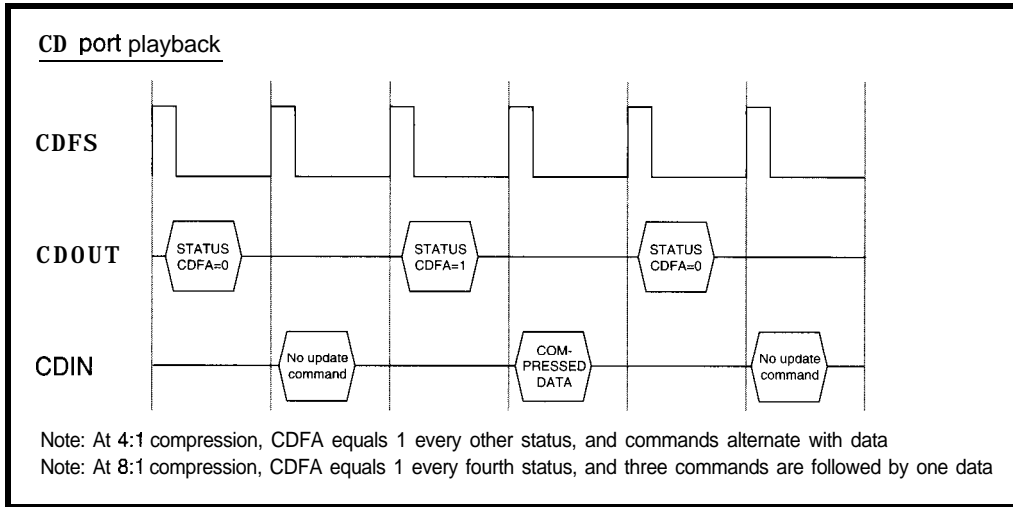
800-SMART-HOME

(800-762-7846)

Hundreds of hard-to-find home automation and wireless control products. Computer control of your home, security systems, surveillance cameras, infra red audio/video control, HVAC, pet care automation, wiring supplies and much more

HOME AUTOMATION SYSTEMS, INC.
 151 Kalmus Dr., Ste. M6, Costa Mesa, CA 92626
 Questions 714-708-0610 Fax 714-708-0614

Figure 10: CDFA set to 1 during playback indicates that the next CD port input frame will be interpreted as a compressed data byte rather than a command byte.



Voic e gets the interrupt routine and timer ready to go, it releases the 2132A reset line and sets CDFS/IRQ*ENA to true. The 2132A comes out of reset and goes into idle mode, and the interrupt routine executes on the next frame sync.

sists when the playback is muted, it is considered valid and the voice playback continues to be muted. allowing the software to act on the DTMF tone.

SOFTWARE

Now that we know just what's involved in operating the 2132A, we can take a closer look at the software that helps make it happen. The detailed interface specification for the low-level software module is summarized in Table 3. The actual assembly language implementation uses the Pascal calling convention. Also provided are a definition module for Modula-2 and a header file for C.

The routines fall into three categories: operational control, buffer handling, and hardware control. In addition to these routines, there is one more very important player hidden just below them and out of view of the application software. That player is the interrupt routine. This routine is set off by the master-frame sync once each frame, and it directly handles communications with the CD port.

OPERATIONAL CONTROL ROUTINES

The first group of operational control routines consists of EnableVoice and DisableVoice. These two perform initialization and shutdown of the system. EnableVoice installs the interrupt routine and initializes the 82C55, 82C54, and 2132-A hardware. It also initializes the global variables used by the software, including all the buffers, and performs the CD-port I/O

synchronization with the help of the interrupt routine. DisableVoice, on the other hand, cleans up the system by shutting down the interrupt hardware and 2132A, uninstalling the interrupt handler, and reinitializing some of the global variables.

To communicate with the CD port, the software must get in sync with the input and output toggling of the port. EnableVoice and the interrupt routine work together to accomplish this. Once Enable

In idle mode, the CD port toggles between input and output with each successive frame. The CDIN line has a pullup on it so the 2132A reads FFH as a command. Since FFH is the No-Update command, the 2132A remains idle until the software gets in sync and starts sending commands. In addition, the status byte output by the port has the CDFA bit set to zero-this is key to the whole process.

<p>Operational Control Routines</p> <p>Start-up/Shut-down Control</p> <ul style="list-style-type: none"> EnableVoice: initializes and enables operation of the 2132A and interrupt routine DisableVoice: disables operation of the 2132A and interrupt routine <p>Mode Control</p> <ul style="list-style-type: none"> PlayEnabled: returns TRUE if the software is set to the Playback mode SetPlayState: sets Playback mode for the software on or off EnablePlay: sets Playback mode on DisablePlay: sets Playback mode off <p>Buffer Handling Routines</p> <p>Record Buffer</p> <ul style="list-style-type: none"> RecDataReady: returns TRUE if data/status have been received and are available RecBufferReset: clears the receive buffer, deleting any data/status bytes in the buffer ReadRecData: returns a data/status byte from the receive buffer if buffer is not empty <p>Playback Buffer</p> <ul style="list-style-type: none"> PlayDataWaiting: returns TRUE if the playback buffer is not empty PlayBufReset: clears the playback buffer, deleting any unsent data bytes SetDefaultPlay: sets the default compressed data byte WritePlayData: places a compressed data byte in the playback buffer <p>Command Buffer</p> <ul style="list-style-type: none"> CommandWaiting: returns TRUE if the command buffer is not empty CommandBufReset: clears the command buffer, deleting any unsent command bytes SetDefaultCommand: sets the default command byte WriteCommand: places a command byte in the command buffer <p>Hardware Control Routines</p> <p>DAA Control</p> <ul style="list-style-type: none"> DAAControl: sets the OFFHK line of the DAA on or off DAAStatus: returns the current status of the OFFHK, RI, and PSQ lines of the DAA

Table 3: A summary of the low-level software interface, describing each of the routines provided.



To get in sync with the 2132A, the interrupt routine reads the CD port on each frame following a reset, and checks for the CDFA bit set to zero. For an input frame, the 2132A is not driving the CDOOUT line, but is being held high by a pull-up resistor. This causes the software to read a 1 for bit 7 and to conclude that the current frame is an input frame.

However, for an output frame, the 2132A outputs a status byte on CDOOUT with the CDFA bit set to 0. The software detects the 0, which signals the current frame as an output frame. The interrupt routine indicates `EnableVoice`, which is waiting for verification on the sync-up. `EnableVoice` then finishes initialization, allowing normal operations to begin on the next frame. The interrupt routine only makes a finite number of attempts at the sync-up, and then reports a failure to `EnableVoice` if the attempt counter reaches zero. This allows `EnableVoice` to clean up and exit gracefully if the sync-up fails, indicating the failure to the application program.

The other operational routines control how the software handles the CDFA bit of the status byte during normal operation. In our earlier discussion of the CD port, we showed how the CDFA bit is handled differently depending on whether the 2132A is in record or playback mode.

In developing the interface for the low-level software, we had a choice of how to handle this moding. We could have made the low-level routines interpret the commands sent to the 21328 and switch to the proper mode based on those commands. This choice adds complexity to the low-level software, but relieves the application software of any concerns with the hardware.

The other option requires the application software to inform the low-level software of which mode to be in. We chose this method to reduce the complexity of the low-level software without placing an undue burden on the application software.

The routines simply modify or read the value of a global variable in the low-level module. This variable is used by the interrupt routine to decide how to handle the CDFA bit of the status byte. These routines do not send any commands to the 2132A, they merely control the operating mode of the software. Sending commands to the 2132A is the responsibility of the application program. Listing 1 provides examples

of how to properly use these routines in an application program in conjunction with commands sent to the 21328 using the command-buffer routines.

RUFFER-HANDLING ROUTINES

There are three groups of buffer-handling routines to manage the flow of compressed data, status, and commands between the 21328 and the application software. Each group of routines represents one of the buffers. The record buffer handles compressed data and status bytes read from the

CD port by the interrupt routine. When the 21328 is in playback, tone-generation, or idle mode, this buffer contains status bytes only.

However, in record mode, the buffer contains interleaved compressed data and status bytes in the same order they are read from the CD port. This requires the application program to exercise some care to ensure that it keeps track of which bytes are status and which are compressed data. For example while in idle mode, as long as you know that you're start-



Let Your Development Fly!

Choose Hitex professional tools for your embedded microprocessor design and get your project off the ground ahead of schedule. Hitex builds all types of microprocessor development tools, from sophisticated in-circuit emulators to remote debuggers, monitors and simulators. Complete solutions are available for:

- ✓ the complete 8051 family
- ✓ 80C86/88, 80C186/188EA/EB/EC/XL, 80286, V20/V30/V40/V50
- ✓ 80386DX/SX/CX/EX
- ✓ 80C165/166/167
- ✓ 68HC11 family
- ✓ 6800x, 6830x, 6833x, 68340

Call Hitex for your free demo package and learn how smooth and efficient development with professional tools really can be!

HITTOOLS Inc.
2055 Gateway Place
Suite 400
San Jose, CA 95110
☎ (408) 451-3986
☎ 1-800-45HITEX
☎ Fax: (408) 441-9486

© Copyright 1991 Hitex. Hitex is a registered trademark of Hitex. All other trademarks are acknowledged by their respective owners.

CALL US NOW!
1-800-45HITEX

hitex

teletest 32

New: TX386
Entry-level 386x
in-circuit
emulator

Listing 1: This sample listing illustrates how to use the low-level software routines to control the DS2132A.

```

CONST
  NoUpdateCmd = 0;
  IdleCmd = OBEH;
  Record8to1Cmd = 23H;
  Playback8to1Cmd = 20H;
  PlaybackOff = FALSE;
  BlankPlaybackData = 0;

(* Initialization *)
DisablePlay;
SetDefaultCommand( NoUpdateCmd );
WriteCommand( IdleCmd , cmdOK );

(* Record Example *)
SetPlayState( PlaybackOff );
WriteCommand( Record8to1Cmd, cmdOK );
REPEAT
  ReadRecData( recdata , rdOK );

UNTIL DoneRecording;
WriteCommand( IdleCmd , cmdOK );

(* Playback Example *)
IF NOT PlayEnabled() THEN EnablePlay END;
SetDefaultPlay( BlankPlaybackData );
WriteCommand( Play8to1Cmd, cmdOK );
REPEAT
  WritePlayData( playdata , wrOK );

UNTIL DonePlaying;
WriteCommand( IdleCmd );
DisablePlay;

```

ing with a status byte, you can check the CDEA bit of each status byte to determine if the next byte is compressed data.

The other two huff&s are similar. They perform the same basic functions for either outgoing compressed data or command bytes. Both handle bytes going from the application software to the CT port. The playback buffer handles compressed data sent from the application to the 2132A for playback. The command buffer performs a similar function for command bytes.

HARDWARE-CONTROL ROUTINES

The DAA has two control inputs and two status outputs in addition to

the analog connections. The OFFHK input controls the phone line; setting it to true takes the telephone line off-hook so calls can be generated or answered. The *RI status output indicates when a ring signal is detected on the line.

The other two signals need a bit more explanation. FCC regulations require that data or recorded-voice calls have a two-second "hilling delay" after the line is taken off-hook and during which the line must be kept quiet. Normal voice calls do not require this delay. The DAA can automatically perform the delay if the *D/V input (D for data, V for voice) is held low. In this case, the DAA squelches the XMIT line for two seconds after being commanded off-hook with the OFFHK input.

Other FCC requirements limit



the output level that the DAA can put on the line during data calls. When *D/V is low, the DAA squelches the XMIT signal if levels are too high. The PSQ output indicates when XMIT is squelched, either for the hilling delay or for exceeding the output levels. The application software should monitor the PSQ signal when in the playback mode and adjust the path gain, if needed.

There are two DAA routines in the low-level software. One is used to control the OFFHK input to the DAA, allowing the application to take the phone line off-hook. The other routine returns the current value of the OFFHK line and the two status outputs, *RI and PSQ. Since the DAA must be set for data to comply with FCC requirements, the *D/V line is not directly controllable. Instead, it is set to the proper value by the EnableVoice routine.

CLOSING COMMENTS

We started the journey toward the "Telephone Wonder Gadget" by putting the 21324 on a PC ISA bus card with enough hardware to hook it up to the telephone network and enough low-level software to take care of most of the messy details in making this complex chip go. Throw in some application software and the basic system is capable of anything from a simple answering machine to a complex voice-messaging system with home-control capability added for good measure.

With a little more hardware on the analog side and the right software, it could serve as a sort of miniPBX with voice menuing and the ability to screen calls. Then, the phone would only ring if the caller has a password (no more telemarketers).

What do you want your Telephone Wonder Gadget to do?

Robert Luzenski develops software and hardware for PCs and embedded systems, including computer communication applications for modems and Internet. Jack Ivey designs hardware and software for embedded systems. Robert and Jack may be reached at robert.luzenski@circellar.com and jack.ivey@circellar.com, respectively.

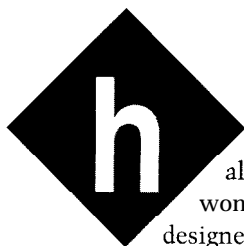
IRS

428 Very Useful
428 Moderately Useful
430 Not Useful

UFO Alert!

SILICON UPDATE

Tom Cantrell



There's a secret for all you would-be wonderous widget designers—the UFOs are

coming!

It might be wise to raise your head from your bench or CRT and take a look around. Oh yes, the UFOs look like your old friends, but that NMI pin is a dead giveaway. Something strange is going on.

The UFO-masters say they are here to serve you. But, as in the old Twilight Zone episode, do they mean to help you with your next design or dish you up for lunch?

Never fear, as your intrepid reporter, I'm ready to dissect these aliens and decide if they're friend or foe.

WHERE NO '51 HAS GONE BEFORE

Lest you think I've turned tabloid (hey, UFO headlines work for them), I should explain that UFO stands for Unidentified Fifty-One and refers to brand new versions of that venerable people's micro, the 8051. Both Intel and Philips have UFOs on the launch pad, and the countdown is starting. This month, we'll take a look at Philips' UFO, which they call XA.

Sure, half a dozen or so suppliers offer more '51 derivatives than you can shake a stick at. But until now, most spinoffs have been created by simply altering I/O functions or boosting the clock rate. The CPU core remains unchanged since its ancient (i.e., late '70s) invention by Intel.

Despite the '51's popularity, it is definitely getting long in the tooth. Not exactly an elegant architecture to

begin with, historic quirks look evermore glaring in the harsh light of competitors' modern offerings.

With the S-bit market rocketing past at 1B units a year, Intel and Philips faced an "ante up or fold 'em" situation with the '51. They either had to significantly upgrade the part or watch it die at the hands of new contenders.

Accepting the challenge means embarking on the primrose upgrade path. Though blazed by other chips of yore, the path still has many forks and obstacles for the unwary.

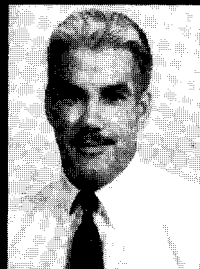
A basic decision at the outset is just what flavor of compatibility to offer. Sure, marketing will sell the new part as compatible no matter what, but there are some serious technical decisions to be made.

One of the most important is whether to preserve object-code compatibility (i.e., whether the chip can run old binaries or whether the source must be reassembled or compiled).

A decision to abandon binary compatibility is not to be taken lightly. First, there's the matter of customers digging through file cabinets and stacks of old floppies to resurrect the source. Then, everyone has to update their tool chests, not just with the new compiler and assembler, but also all the other stuff—emulators, debuggers, monitors, and so on. Finally, the updated software is likely to need retuning either to take advantage of new features or to deal with timing differences between the old and new CPUs.

However, the latter issue of retuning is also an argument for abandoning the past. PC programmers have learned to insulate their programs from CPU speed differences and, on the desktop, the goal is to do things faster anyway.

But, embedded control programs are a different story. First, even if an effort is made (often not or only half-heartedly) to write timing-independent software, it's almost invariable that a few gotchas will pop up. Second, unlike the PC, unconstrained application speed up is not necessarily good. Nobody complains if their spreadsheet



The old 8051, despite its popularity,

is losing ground to current competition. Philips is fighting back. Their new "8051s" wear the same 8-bit jacket, but underneath lies a 16-bit ALU, register set, and bus interface.

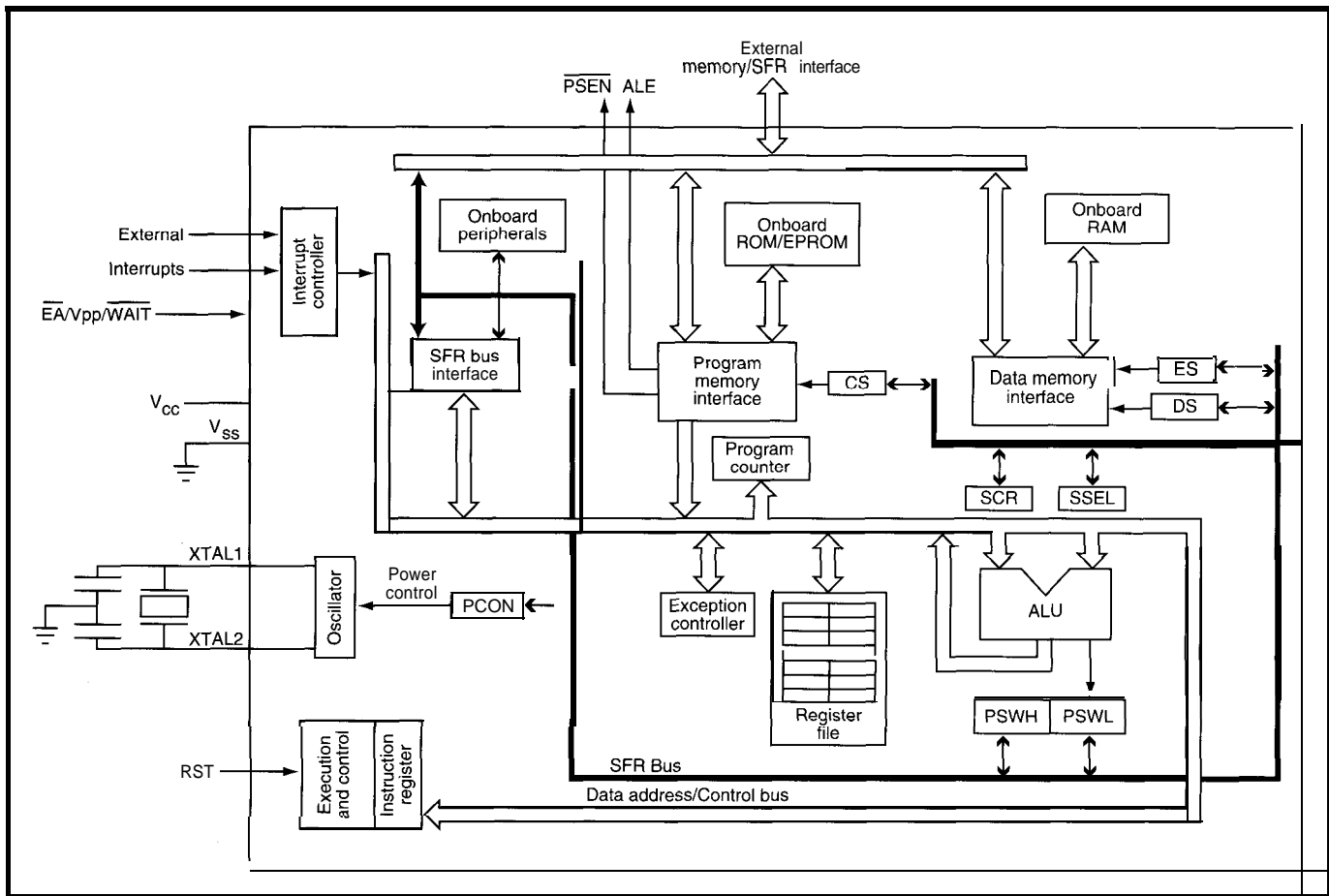


Figure 1--Though not as obvious as a *third eye*, the 16-bit ALU, extended PSW, and segment registers (CS, ES, and DS) distinguish the XA from the '51 it impersonates.

recalculates 50% faster, but how about a turbocharged pacemaker? Sounds like a rush for the patient-to a lawyer, that is.

The argument that the source has to be modified in any case helps make the decision to foresake binary compatibility less daunting, but it isn't pivotal. The main reason to move onward is that it is very difficult to make a lot of progress if you're saddled down with old baggage.

Let's follow the path chosen by the XA and see where it leads. Along the way, we'll see how it avoids the hazards and dead ends that tripped up the original '51.

INVASION OF THE CHIP SNATCHERS

The UFOs try to pass themselves off as regular 8-bit chips, but scratch beneath a thin marketing veneer and you'll see a 16-bit ALU, register set, and bus interface [see Figure 1).

Looking further, it quickly becomes apparent that the XA is a '51

in little more than name. The XA does not share binary or even assembly-language-source compatibility with the '51. Instead, '51 assembly source must be translated to XA source and then reassembled.

Those of you who remember the dubious record of previous translators (notably the 8080 to 8086 translators

offered at the dawn of the PC age) have a right to be concerned. You remember how programs would expand and slow down with lots of weird instructions inserted hither and yon to scramble flag bits and translate odd opcodes.

Thankfully, the XA translation scheme appears much cleaner. The XA adopts a '51 superset mentality in

Mnemonic	Usaae
MOV, MOV, MOV, MOV, LEA, XCH, PUSH, POP, PUSHU, POPU	Data Movement
ADD, ADDS, ADDC, SUB, SUBB	Add and Subtract
MULU.b, MULU.w, MUL.w, DIVU.b, DIVU.w, DIVU.d, DIV.w, DIV.d	Multiply and Divide
RR, RRC, RL, RLC, LSR, ASR, ASL, NORM CLR, SETB, MOV, ANL, ORL	Shifts and Rotates Bit Operations
JB, JBC, JNB, JNZ, JZ, DJNZ, CJNE	Conditional Jumps and Calls
BOV, BNV, BPL, BCC, BCS, BEQ, BNE, BG, BGE, BGT, BL, BLE, BLT, BMI	Conditional Branches
AND, OR, XOR	Boolean Functions
JMP, FJMP, CALL, FCALL, BR	Unconditional Jumps, Calls, and Branches
RET, RETI	Return from subroutines and interrupts
SXT, NEG, CPL, DA	Sign Extend, Negate, Compl., Decimal Adj.
BKPT, TRAP#, RESET	Exceptions
NOP	No Operation

Table 1--The XA instruction set is a superset of the '51

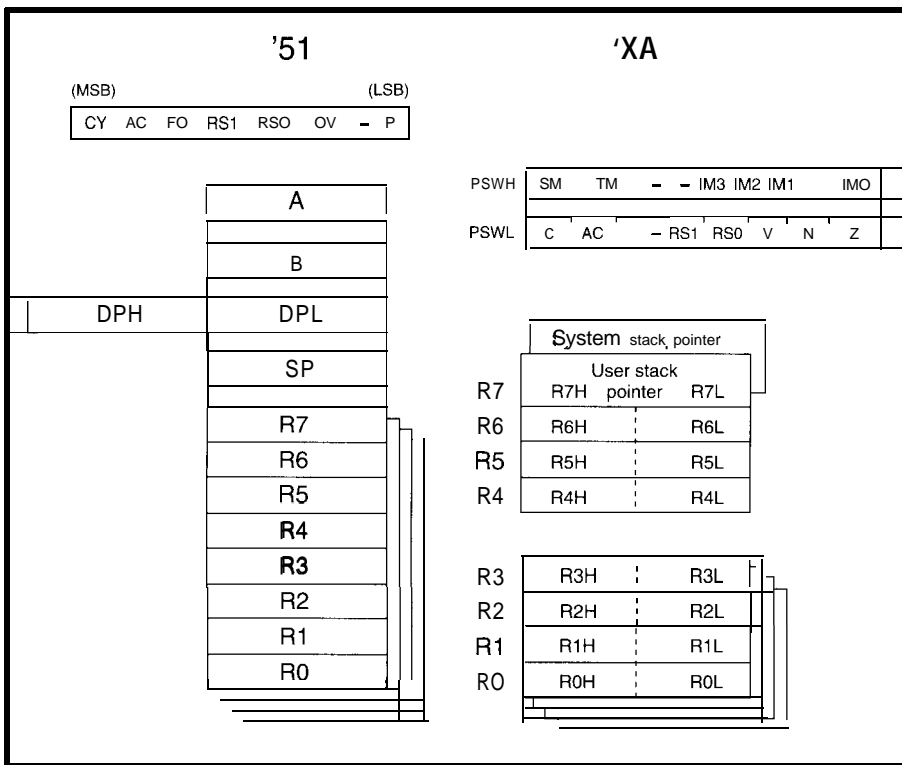


Figure 2—The XA eliminates the '51 accumulator, data-pointer, and stack-pointer bottlenecks

which instructions, registers, memory, flags, and so on encompass their '51 counterparts, making conversion straightforward. Notably, almost all '51 instructions translate 1 for 1 to XA instructions (the XA instruction set is shown in Table 1). The only exception is the rarely used XC HD (a 4-bit nybble swapper), which must be replaced with a multiinstruction sequence.

While the debate over instruction sets is never ending, I think most agree that fast instructions are better than slow ones. The '51's leisurely performance (a whopping 12 clocks per instruction) is boosted by a factor of 3 to 4 times in the XA.

Figure 2 compares the XA and '51 register sets. Right off the bat you'll notice that the registers are 16 bits wide rather than 8 bits as in the '51. The eight registers are byte (low and high) or word addressable. In fact, for some operations (shifts, multiplies, and divides) certain register pairs (R0/1, R2/3, R4/5, and R6/7) can even be accessed as 32 bits. As in the '51, four banks of registers are provided.

With a general-purpose register set, the XA dispenses with the '51's dreaded accumulator (A&B) and memory (DPTR) bottlenecks. Speaking

of memory bottlenecks, the 8-bit stack pointer (SP) of the '51 is extended to 16 bits in the XA. Larger stack space, not to mention the ability to easily access and manipulate it (i.e., R7 can be used as an index register) should help ease the pain of long-suffering '51 compiler writers.

Indeed, the XA offers two stack pointers in support of a new user- and system-mode protection scheme. Exceptions push the state onto the system stack, leaving the user stack free for application software. Note that the stack on the XA grows down (like almost every other CPU) instead of up as on the '51.

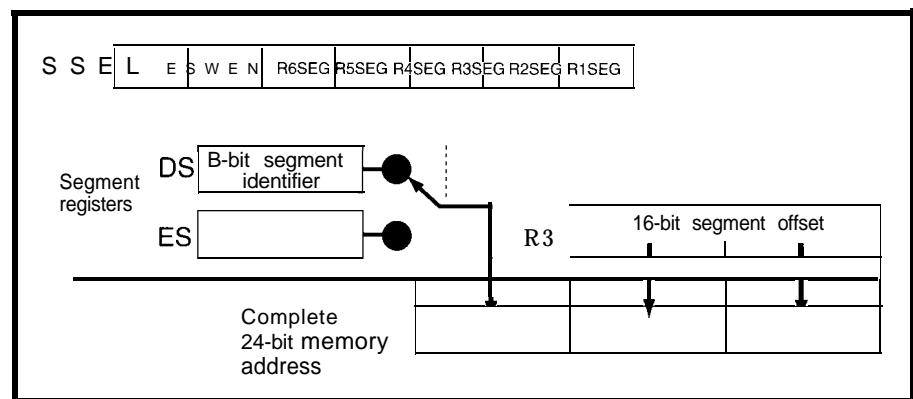


Figure 3—Unlike the x86, the XA segment scheme appends (not adds) the segment to the 16-bit offset. Segment references are assigned to registers (via SSEL), rather than implied by instructions.

The XA extends the byte-wide '51 PSW to 16 bits (PSWH and PSWL). The lower half corresponds closely to the '51 with matching and auxiliary carry, overflow, and register-bank select flags. The XA dispatches with the '51's general-purpose flags (PSW.5 and PSW.1) and parity (P) flag in favor of an N (negative sign bit) and Z (zero) flags. To avoid flag shuffling, a '51-compatible version of PSWL is made available for backward compatibility.

The '51 got by without a Z flag by performing compare-and-branch functions in a single operation, which makes sense given that only the accumulator (ACC) could be compared against it. Since the XA dispenses with the accumulator bottleneck altogether (i.e., you can compare lots of different things, not just the accumulator), a Z bit was called for.

The upper half of the XA PSW contains the user and supervisor bit (the PSWH can only be accessed in supervisor mode), a trace bit (causes an exception after each instruction—good for a debugger), and four bits that define the level of the current task-supporting software (and I imagine in the future, hardware) prioritization.

LOST IN ADDRESS SPACE

Having dealt ably with the '51's programming singularities, the XA designers turned their attention to the "64K problem." Actually, on the '51, it's a "64K code + 64K data" problem. However, since many designs overlap the two spaces, it's still a problem (128K isn't enough either).

Being about the last 64K chip in the world to face the issue, the XA

designers were able to learn from the good, bad, and ugly of previous approaches. The resulting segment scheme relies on code, data, and extra (CS, DS, and ES) segment registers to boost address space to 16 MB (24 bits).

The use of the word "segment," not to mention the naming convention (CS, etc.), is likely causing distress for those of you who haven't yet learned to love the similarly nomenclatured 'x86 scheme. Lest you contemplate suicide by soldering iron, I'm pleased to report that the XA scheme is really quite simple and effective. As in the 'x86, the segment registers point a 16-bit address into the larger address space, but the similarity ends there.

As shown in Figure 3, note how the 8-bit segment register contents are merely appended to the front of the 16-bit address as opposed to the shift&add of the 'x86. Besides alleviating the confusion of keeping track of where you are, a notable byproduct is that a given physical memory location is accessible via one, and only one, segment value.

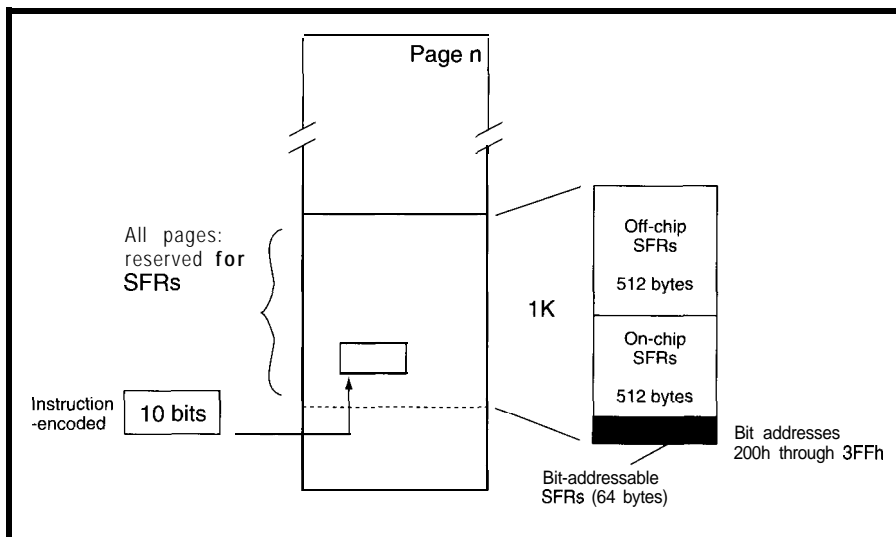


Figure 4— The SFR space, bit- and byte-addressable as in the '51, is replicated in each 64-KB page for speedy access.

For example, address 123456H can only be addressed if the segment register contains 12H. In conjunction with the system and user protection scheme (the segment registers can only be programmed in system mode), it provides a fairly bulletproof way to keep tasks from interfering with each other.

The worst part about the 'x86 scheme is the way segment-register usage is implied by instructions (loosely and arbitrarily, critics would say). Not to worry, though, since if you don't like (or can't remember) the implication, just use a segment-override prefix (subject, of course, to its own arcane rules and restrictions).

BCC52 BASIC-52 COMPUTER/CONTROLLER

The BCC52 controller continues to be Micromint's best selling single-board computer. Its cost-effective architecture needs only a power supply and terminal to become a complete development system or single-board solution in an end-use system. The BCC52 is programmable in BASIC-52, (a fast, full floating point interpreted BASIC), or assembly language.

The BCC52 contains five RAM/ROM sockets, an "intelligent" 2764/128 EPROM programmer, three 8-bit parallel ports, an auto-baud rate detect serial console port, a serial printer port, and much more.

PROCESSOR

- 80C528-bit CMOS processor w/BASIC-52
- Three 16-bit counter/timers
- Six interrupts
- *Much more!

Input/Output

- Console RS232 - autobaud detect
- Line printer RS-232
- Three 8-bit parallel ports
- EXPANDABLE!
- Compatible with 12 BCC expansion boards

MEMORY

- 48K RAM/ROM, expandable
- Five on-board memory sockets
- Either 8K or 16K EPROM

To Order Call 1-800-635-3355

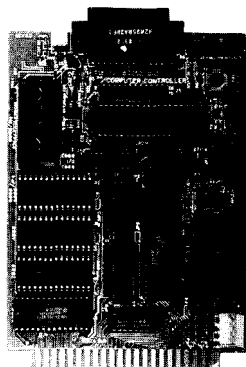
TEL: (203) 871-6170

FAX: (203) 872-2204

BCC52	Controller board with BASIC-52 and 8K RAM	\$1 89.00	Single Qty.
BCC52C	Low-power CMOS version of the BCC52	\$199.00	
BCC52I	-40°C to +85°C industrial temperature version	\$294.00	
BCC52CX	Low-power CMOS, expanded BCC52 w/32K RAM	\$259.00	

CALL FOR OEM PRICING

MICROMINT, INC. 4 Park Street, Vernon, CT 06066
in Europe: (44)0285-658122 • in Canada: (514)336-9426 • in Australia: (3) 467.7194
Distributor Inquiries Welcome!



Smart Little Genius™ Controller

Starting at \$119, Qty 1

Our C-programmable miniature controllers are ideal as the brains for control applications, data acquisition, and test and measurement. Features include digital I/O to 400 lines, ADCS, DACS, relays, solenoid drivers, RS232/RS485, battery-backed RAM, clock, watchdog, LCDs, keypads, enclosures and more. Use our simple, yet powerful, Dynamic C™ development system (\$195 integrated editor, compiler and debugger) for quick project completion!

24-Hour AutoFax
916.753.0618. Call
from your FAX.
Request catalog 18.

1724 Picasso Ave.
Davis, CA 95616
916.757.3737
916.753.5141 FAX



By contrast, the XA ties segments to registers, not instructions, and lets the programmer explicitly make the assignment. A programmable configuration register (SSEL) defines which segment is to be associated with each register (R0-R6; R7, the stack pointer, is always referenced via DS). Thus, segment selection has nothing to do with which instruction is executing, only which register is being accessed.

The XA, like the '51, defines SFRs (Special Function Registers) as the mechanism to access control and status registers, I/O ports, and so on. Also like the '51, these are mapped into a directly addressable (and only directly addressable) block of the address space. For instant accessibility at all times, the XA 1-KB SFR space (boosted from 128 bytes in the '51) is replicated in each 64-KB bank (see Figure 4).

Note the interesting provision for off-chip SFRs which would seem to support coprocessorlike connection to internal hardware as well as a no-muss, no-fuss way to migrate an

Figure 5—With the registers and on-chip RAM and SFRs, 128 bytes are bit accessible. The XA architecture allows for 16 registers (R0-R15) though only 8 (R0-R7 needed for compatibility) may be offered on a particular chip.

Bit space		Overlaps bytes..	
Start	End	Type	Start End
0	←→ 0FFh	Registers	R0 ←→ R15
100h	←→ 1FFh	Direct RAM	2 0 h - 3 F h
200h	←→ 3FFh	On-chip SFRs	400h ←→ 43Fh

external peripheral function onto a higher integration derivative.

A popular feature, retained from the '51 (no choice really, given the translatability constraint), is bit addressing. The 1024 bit addresses (like SFRs, a factor of eight expansion over the '51) are mapped into the register file, on-chip RAM, and SFRs (see Figure 5). So, bit-banging these hot spots is quick and easy.

WE INTERRUPT THIS PROGRAM

The '51's somewhat feeble interrupt scheme has been put out of its misery in the XA.

As shown in Figure 6, the XA defines a 64-entry (256-byte) vector table which specifies a handler address

and initial PSW. Note that the 16-bit handler address requires all handlers to be located so they start in the first 64 KB of memory (i.e., CS=0).

The event interrupts come from on-chip peripherals or external pins and cover the entire subject as far as a '51 is concerned.

The XA goes further by defining a TRAP (O-15) instruction which is a handy way to implement an RTOS call since it provides a way (the only way) for user software to request system-level protected services.

Finally, an exception mechanism is provided to deal with gotchas like divide by 0, stack overflow, the previously mentioned Trace exception, and so on. One notable improvement is Exception 16 (the highest priority) which is NMI. Yes, the XA has a real nonmaskable interrupt which, unlike the '51, doesn't depend on trusted software (an oxymoron, yes?) to remain diligent.

Whatever the source, in response to an interrupt, the XA stacks 6 bytes of information on the system stack as shown in Figure 7. This is quite different than the '51, which pushes only 16 bits of PC. The XA designers had to automatically stack PSW to make the protection and trace stuff work.

A similar 16- versus 24-bit question concerns the size of the PC address pushed and popped for calls and returns. Small and/or translated programs may prefer to see 16-bit addresses as on the '51 while new, larger applications want all 24 bits. The XA designers decided the best choice was not to make a choice. So, they put in a configuration bit that lets you have it your way.

DOWN TO EARTH

Perhaps to avoid architecture shock among loyal '51 customers, the

e WORLD'S SMALLEST

86™

DX

Embedded PC with
Floating Point,
Ethernet & Super
VGA Only 4" x 4"

4

The PC/II +i includes:

- 486DX™ CPU at 25MHz or 33MHz clock frequency
- Full 8K Cache with Floating Point
- Ethernet Local Area Network
- Local Bus Super VGA Video/LCD
- Up to 2MBytes Flash™ with TFFS
- 4 or 16MBytes User DRAM
- PC/104 or ISA Bus compatible option (with adapter)
- 4" x 4" Format; 6 watts power consumption at +5 volt only



(416) 245-2953

megatec®

486DX and Flash are registered trademarks of Intel Corp. as are PC, AT or IBM, megatec of Megatec Computer (1986) Corp

125 Wendell Ave. • Weston, Ont. • M9N3K9 • Fax. (416) 245-6505

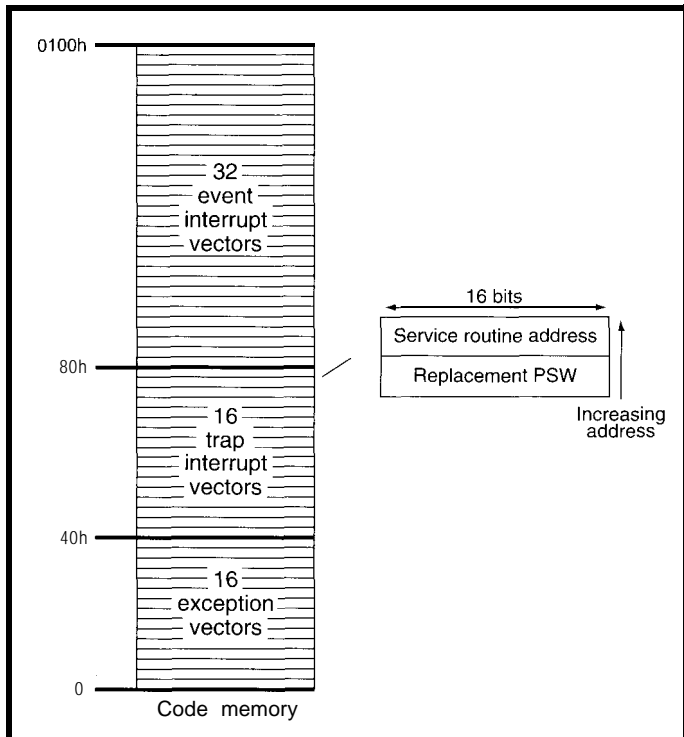


Figure 6—The XA adopts a table-based vector scheme supporting up to 64 interrupts including I/O events, TRAP instructions, and exceptions.

XA presents a deceptively familiar face to the outside world.

The comforting complement of standard '51 on-chip I/O (UART, timers, etc.) remains largely unchanged, though there are some helpful upgrades. The UART now offers error detection (framing, overrun, etc.) while the timers are upgraded with programmable timebase (CPU clock divided by 4, 16, or 64). The I/O ports supplement the '51's quasi-bidirectional mode with push-pull, open-collector, and high-impedance options. In general, the changes are software transparent and likely call for only minor programmer attention.

The bus interface is equally customary, featuring well-known '51 signals like ALE, PSEN*, RD*, WR*, EA*, and so on. One new addition is WRH* (Write High), which is used to

write the upper byte of the data bus when the XA is in 16-bit bus mode. Note that an RDH* signal isn't needed since the XA ignores the unnecessary byte when making a byte access to a 16-bit bus.

While the XA can freely access bytes in either 8- or 16-bit bus mode, 16-bit word accesses must start at an even address in either mode. If you were wondering, this explains why the 24-bit address is padded to 32 bits when stacked in response to an interrupt.

The XA also retains the familiar multiplexed address and data bus of the '51 with a slight twist. Figure 8 shows a typical (at first glance) connection to an 8-bit peripheral. But, note how the data is multiplexed starting at A4, leaving the lower four address lines demultiplexed. The

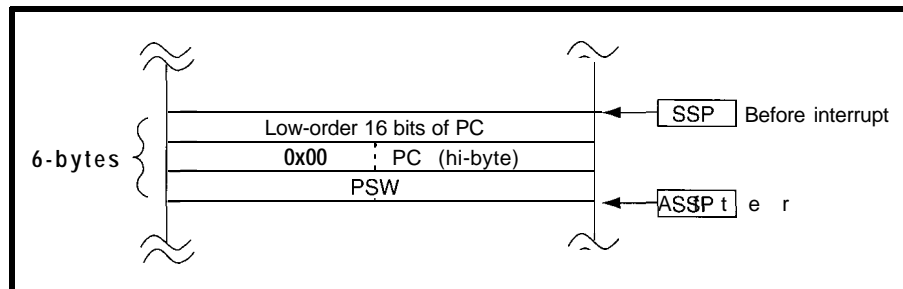
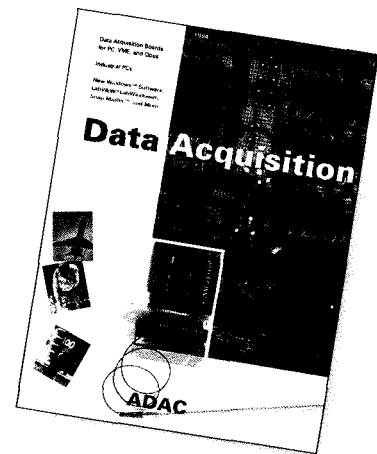


Figure 7—While the '51 stacks only a 16-bit PC in response to an interrupt, the XA also stacks the high PC and 2-byte PSW. Note that padding the high PC preserves word alignment.

NEW Data Acquisition Catalog

Covers expanded low cost line.



FREE!

NEW 120 page catalog for PC, VME, and Qbus data acquisition. Plus informative application notes regarding anti-alias filtering, signal conditioning, and more.

NEW Software:

LabVIEW®, **LabWindows®**, **Snap-Master™**, and more

NEW Low Cost I/O Boards

NEW Industrial PCs

NEW Isolated Analog and Digital Industrial I/O

New from the inventors of plug-in data acquisition.

Call, fax, or mail for your free copy today.

ADAC

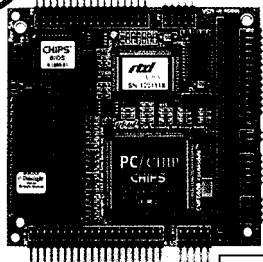
American Data Acquisition Corporation

70 Tower Office Park, Woburn, MA 01801

Phone: (800) 648-6589 Fax: (617) 938-6553

Replace Four Conventional PC/104 Modules with One SuperXT™ CMF8680 cpuModule™

Embedded PC/XT Controller with
Intelligent Power Management



PC/104 Compliant
Actual Size: 3.6 x 3.8 x 0.6"

\$449

100 pcs

- PC/XT compatibility with 286 emulation
- 14 MHz, 16-bit 8086 CPU
- +5V only; 1.6W at 14.3 MHz, 1W at 7.2 MHz
- Intelligent sleep modes, 0.1 W in Suspend
- ROM-DOS and RTD enhanced BIOS
- Compatible with MS-DOS & real-time operating systems
- 1 M bootable Solid State Disk & free software
- 4K-bit configuration EEPROM (2K for user)
- 2M on-board DRAM
- IDE & floppy interfaces
- CGA CRT/LCD controller
- Two RS-232 ports, one RS-485 port
- Parallel, XT keyboard & speaker ports
- Optional X-Y keypad scanning/PCMCIA interface
- Watchdog timer & real-time clock

Expand This Or Any PC/104 System
with the

CM106 Super VGA Controller utilityModule™

- Mono/color STN & TFT flat panel support
- Simultaneous CRT & LCD operation
- Resolution to 1024 x 768 pixels
- Displays up to 256 colors

\$223

100 pcs

Speed Product Development with the
DS8680 Development System

Your DS8680 includes the CMF8680, CM102 keypad scanning/PCMCIA, CM104 with 1.8" 815MB hard drive, CM106 SVGA controller & CM5406 12-bit, 100 kHz dataModule™ in an enclosure with external power supply, 3.5" floppy, keyboard, keypad, TB50 terminal board, SIGNAL*VIEW™, SIGNAL*MATH™, MS-DOS, SSD software & rtdLinx™ for just
\$2950.

For more information on our PC/104 and
ISA bus products, call today.



Real Time Devices USA

2200 Innovation Blvd. • P.O. Box 906
State College, PA 16803 USA
(814) 234-8087 / Fax: (814) 234-5218

RTD Europa • RTD Scandinavia

Real Time Devices is a founder of the PC1104 Consortium

primary intention is to support high-speed burst access (e.g., DRAM page mode) of up to 16 sequential bytes of code without requiring an ALE cycle. It also means that an address latch may not be required if the only external ICs are simple peripherals since they typically demand just a couple of address lines.

Another quirk of the '51 was the lack of a WAIT line. Maybe at the time, the designers were safe in assuming other chips would have no trouble keeping up. Unfortunately, that decision haunts '51-derivative suppliers to this day, serving as pretty much a show

stopper to the otherwise simple idea of boosting the clock rate. Ironically, while adding the WAIT line, the XA designers largely eliminated the need for it by including an on-chip wait and bus-cycle timing generator.

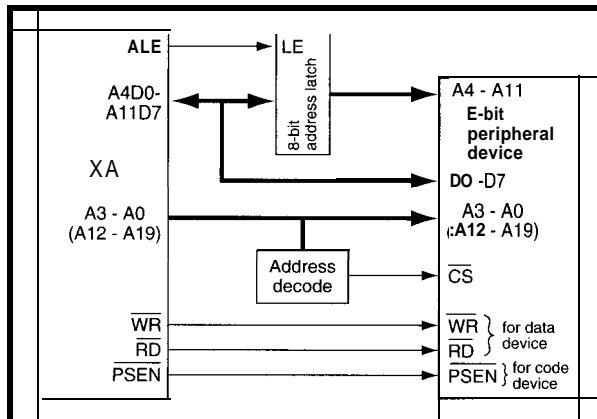


Figure 8—The XA bus interface is quite like the '51, except the lowest address bits aren't multiplexed.

Listing 1—Existing '51 code easily converts to the XA. Notice the assignment of '51 registers (e.g., A, B) to XA registers (e.g., R4L, R4H) for emulation purposes.

```
;StepCal calculates a trip point value for motor movement based
;on a percent of pointer full scale (0-100%)
;Call with target value in A. Returns result in A and StepResult.

StepCal: MOV Temp2,A           ;Save step target for later use
         MOV B,#StepLow       ;Get low byte of step increment
         MUL AB               ;Multiply this by the step target
         MOV StepResult,B     ;Save high byte as partial result
         MOV Temp1,A         ;Save low byte to use for rounding

         MOV A,Temp2         ;Get back the step target
         MOV B,#StepHigh     ;Get high byte of step increment
         MUL AB              ;and multiply the two

         ADD A,StepResult    ;Add the two partial results
         JNB Temp1.7,Exit    ;Least significant. byte > 80h?
         INC A               ;If so, round up the final result
Exit:    ADD A,#MotorBot     ;Add in the 0 step displacement
         MOV StepResult,A    ;Save final step target
         RET
```

```
StepCal: MOV Temp2,R4L       ;Save step target for later use
         MOV R4L,#StepLow   ;Get low byte of step increment
         MULU.b R4,R4H      ;Multiply this by the step target
         MOV StepResult,R4H ;Save high byte as partial result
         MOV Temp1,R4L     ;Save low byte to use for rounding

         MOV R4L,Temp2     ;Get back the step target
         MOV R4H,#StepHigh ;Get high byte of step increment
         MUL R4,R4H        ;and multiply the two

         ADD R4L,StepResult ;Add the two partial results
         JNB Temp1.7,Exit  ;Least significant byte > 80h?
         INC R4L,#1        ;If so, round up the final result
Exit:    ADD R4L,#MotorBot  ;Add in the 0 step displacement.
         MOV StepResult,R4 ;Save final step target
         RET
```

BACK TO THE FUTURE

The XA claim of '5 1 compatibility is arguably credible. As shown in Listing 1 and Table 2, '51 code translates reasonably. Sure, there's some code expansion (note the NOP insertions since branch targets must be word aligned), but it's more than offset by a nearly four-times increase in speed.

While a code fragment looks nice, I caution that a translation exercise can get tricky deep in the bowels of a bizarre program. Besides the previously mentioned timing differences, there are a whole host of gotchas to watch out for.

For instance, the change in stack formats is likely to trip up code that indirectly (i.e., not via PUSH and POP) messes with the stack. Meanwhile, the instruction-size difference will wreak havoc with programs that rely on instructions to fit in a certain area or branches to have a certain reach (a jump table might have both problems). Also, watch out for PC-relative accesses (e.g., @MOV@A+PC) since the XA's PC likely won't be pointing to the same place the '5 1's PC does.

Your choice with the XA is to translate old programs or write new ones, but not both. I suppose it would be possible to try to mix-and-match

'5 1 and XA code, but I suspect it's very ugly, if not impossible. Why not just bite the bullet and go all the way with XA? Thanks to the easy programmer's model, high performance, and the familiar-yet-improved bus and I/O, I suspect most '5 1 users will welcome a close encounter with this UFO. □

Tom Cantrell has been an engineer in Silicon Valley for more than ten years working on chip, board and systems design and marketing. He can be reached at (510) 657-0264 or by fax at (510) 657-5441.

Statistic	80C51 code	XA translation	Comments
Code bytes	28	40	one NOP added for branch alignment on XA
Clocks to execute	300	78	includes XA prefetch queue analysis, raw execution is 66 clocks
Time to execute @ 20 MHz	15 μ s	3.9 μ s	a nearly 4-times improvement without any optimization

Table 2—The XA executes the routine very quickly, even though the amount of code does grow slightly in translation.

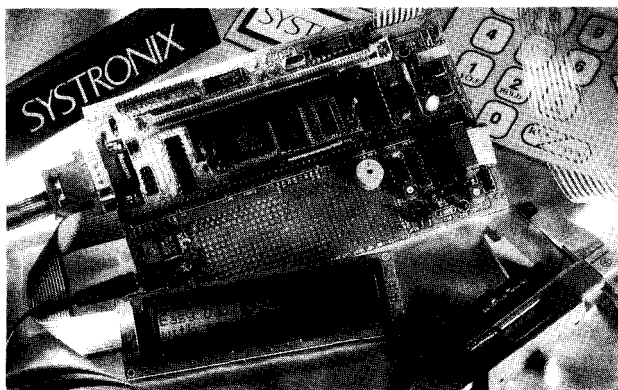
CONTACT

Philips Semiconductors
8 11 East Arques Ave.
Sunnyvale, CA 94088-3409
Attn: Mike Thompson
(408) 991-5207

IRS

431 Very Useful
432 Moderately Useful
433 Not Useful

NEW! UNIVERSAL DALLAS DEVELOPMENT SYSTEM from \$199!

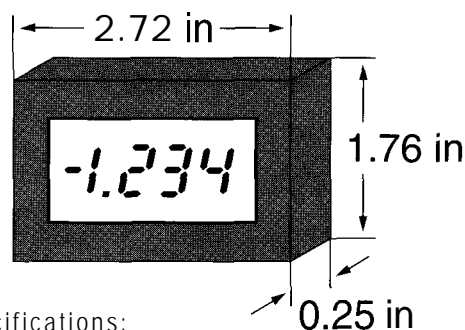


- It's a complete 8051-family single board computer!
- One board accommodates any 40 DIP DS5000, 40 SIMM DS2250, 40 SIMM DS2252, or 72 SIMM DS2251, 8051 superset processor! Snap one out, snap another in.
- Programs via PC serial port. Program lock & encrypt.
- LCD interface, keypad decoder, RS232 serial port, 8-bit ADC, four relay driver outputs, four buffered inputs.
- Power with 5VDC regulated or 6-13 VDC unregulated
- Large prototyping area, processor pins routed to headers
- Optional enclosures, keypads, LCDs, everything you need
- BCI51 Pro BASIC Compiler w/50+ Dallas keywords \$399

SYSTRONIX® TEL: 801. 534. 1017 FAX: 801. 534. 1019
555 South 300 East, Salt Lake City, UT, USA 84111

3% DIGIT LCD PANEL METER

-Available now at an unheard of price of \$15 plus s & h
New! Not surplus!



Specifications:

Maximum input: ± 199.9 mV
additional ranges provided through external resistor dividers
Display: 3½-digit LCD, 0.5 in. figure height, jumper-selectable decimal point
Conversion: Dual slope conversion, 2-3 readings per sec.
Input Impedance: > 1 OOM ohm
Power: 9-12 VDC @ 1 mA DC

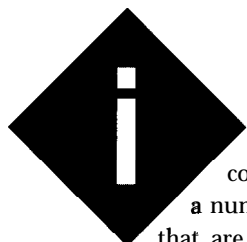
Circuit Cellar, Inc.

4 Park Street, Suite 12, Vernon, CT 06066
Tel: (203) 875-2751 Fax: (203) 872-2204

Micros Behind Bars

EMBEDDED TECHNIQUES

John Dybowski



In last month's column, I looked at a number of media that are commonly

employed in the field of Auto ID with a special emphasis on bar code. I touched on everything from the giant bar codes on rail cars, which move past xenon scanners, to two-dimensional wonders, which look more like artwork than encoded information. The range of complexity spanning the various symbologies collectively called bar codes is quite expansive.

And, as I pointed out last time, that range of complexity hinges on the fact that the industry is centered primarily around economic rather than technological concerns. Because of this practical focus, many of the older, simpler symbologies are still used heavily to this day. Codes such as

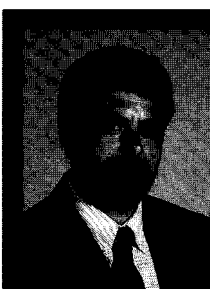
Interleaved Two of Five exist side by side with such fiendishly complex multidimensional representations as VeriCode. Simply put, these older codes are kept around because they still serve their purpose well.

With the emphasis on technology being especially strong in the computer field, it's too easy to forget what pays the bills. Auto ID represents many technical fields pressed to serve the industrial and financial sectors. The bottom line is results, and many of these applications do just fine with a moderate dose of technology.

To those technology zealots who question how processors like the 8051 and 6805 not only survive but prosper, the answer is simple. They reliably provide useful services at low cost. In fact, 8051-class processors offer more performance than is needed for many applications. Bar-code readers are an example of this type of commodity.

CODE 39

Code 39 is a bar-code symbology with a full alphanumeric character set. A unique start/stop code { * } and seven special characters (- \$ / + % and space) are also included in the character set. The name 39 is derived from its code structure of three wide elements out of a total of nine. These nine elements are composed of five bars and four spaces.



To gain a better understanding of how bar code works, John picks one apart. He then looks at the hardware and firmware required to get a real microcontroller decoding real bar code.

Char.	Pattern	Bars	Spaces	Char.	Pattern	Bars	Spaces
1		10001	0100	M		11000	0001
2		10010	0100	N		00101	0001
3		11000	0100	O		10100	0001
4		00101	0100	P		01100	0001
5		10100	0100	Q		00011	0001
6		01100	0100	R		10010	0001
7		00011	0100	S		01010	0001
8		10010	0100	T		00110	0001
9		01010	0100	U		10001	1000
0		00110	0100	V		01001	1000
A		10001	0010	W		11000	1000
B		01001	0010	X		00101	1000
C		11000	0010	Y		10100	1000
D		00101	0010	Z		01100	1000
E		10100	0010	-		00011	1000
F		01100	0010	.		10010	1000
G		00011	0010	SPACE		01010	1000
H		10010	0010	*		00110	1000
I		01010	0010	\$		00000	1110
J		00110	0010	/		00000	1101
K		10001	0001	+		00000	1011
L		01001	0001	%		00000	0111

Table 1--The Code 39 encodable character set consists of 10 numeric digits, 26 alphabetic characters, and 8 special characters.

Unlike some of the other more awkward bar codes, Code 39 uses only two element widths. These are usually simply described as narrow and wide. Using the normal convention, a narrow bar or narrow space is called the *x dimension*. All *x* dimensions must be of equal size within the symbol. The dimension of wide bars and spaces is a multiple of *x*. This ratio can vary within certain proportional limits but, once selected, must remain consistent throughout the symbol. Generally, a wide-to-narrow ratio in the range of 2:1 to 3:1 is acceptable for most Code-39 symbols.

The combination of narrow and wide elements in a Code-39 character always consists of six narrow and three wide elements. A space is included between characters as a separator. No information is contained in the space; it functions only to delimit the characters from each other.

A special code (an ASCII *) is defined as a start/stop character. The purpose of this code is to identify the leading and trailing ends of a bar-code symbol. The bar-space pattern of this code is unique and allows the symbol to be bidirectionally scanned.

Table 1 shows the Code-39 character assignments for all available codes. Note how the last four codes in the table "don't fit" the established coding pattern. Interestingly, if you take away these nonconforming characters you end up with 39 characters. Rumor has it that these 39 characters composed the original character set and are the basis for the Code-39 name. Whatever the case, Figure 1 offers an example of how to decode a Code-39 character "A".

Code 39 is classified as a discrete code since each encoded character is capable of standing alone. That is, the intercharacter space (or gap) is not considered an integral part of the character code and, as a result, enjoys somewhat loose tolerances. The

minimum intercharacter width is the *x* dimension and the maximum is 3*x*.

Combining the desirable discrete attribute with a fixed structure (3 wide elements out of 9) results in code that is classified as self-checking. With this feature, the possibility of a missed decode is much less likely since a substitution error can only occur if two or more elements are misinterpreted. This could happen, for ex-

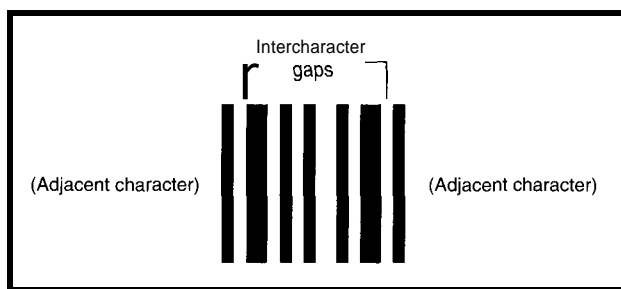


Figure 1—Each Code-39 character is represented by five bars and four intervening spaces. This symbol represents the character "A".

ample, if a spot on a narrow bar lined up with a void on a wide bar and the resulting pattern turned out to be a legal-character depiction.

Another benefit of discrete codes is that they are well matched to certain printing processes. Some types of printers can maintain very tight resolution between elements within a character but are unable to maintain such accuracy in the space between characters. Obviously, these printers are fixed-font devices in which each character code is fully formed. This ensures that tolerances are held tightly within each character. The space between characters is dependent on the printer's mechanical motion and therefore less precise.

In addition to the bar-space pattern that makes up a bar-code charac-

ter and the intercharacter gaps that delimit these characters, there is one more component to a bar-code label. Bar code must be framed with areas free of any printing on either side of the "picket fence" pattern. This region is referred to as the *quiet zone*.

Now, with this information we can take the pattern of ones and zeros to assemble a start code, some data characters, and a stop code. Framing this with the requisite quiet zones results in a standard bar-code label. These elements are depicted in Figure 2.

HAND SCANNING

Many methods exist for converting a bar code's optical information to an electrical form suitable for input into a computer. In all cases, the printed pattern of bars and spaces is converted into a binary bitstream as it is scanned physically or by purely electrical means. Since this data is transformed into the time domain, the bar-code processor must proceed by first recording timing information relative to each bar or space event.

Although some autoscanning readers are very accurate in their initial and absolute scan velocities, this is not a requirement. The main feature these devices offer is their rapid repetitive scanning action. Combined with a slight dither of the light source, the same symbol can be scanned numerous times through slightly different paths until a good read is recorded. This multiple scanning illustrates the data redundancy that is built into the vertical dimension of a bar code.

This redundant data can be used with a handheld scanner as well. In the event of a decode failure, the natural inclination is to scan the label again. In this case, it is highly unlikely that the same part of the label will be scanned a second time.

Some applications require the use of noncontact automatic scanners.

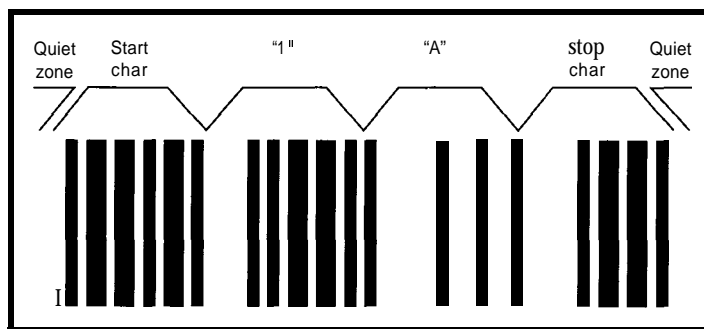


Figure 2—Quiet zones, start/stop codes, and data codes constitute the elements of a bar code. The encoded information here is "1A".

The two-dimensional bar codes I described last month certainly demand this caliber of performance. More conventional bar codes may also dictate the use of such sophisticated devices as well. For instance, more complex devices must be used for tracking materials on rapidly moving conveyer belts, high-volume, point-of-sale operations, and long-range, point-and-shoot warehouse applications. Since this is a field in which high and low tech coexist side by side, dealing with conventional bar code in unique situations is possible.

Low-tech devices usually refer to hand-held bar-code wands. Of course, this distinction is purely relative and does not imply that such devices suffer from a lack of technological elegance. The fact is, until recently, coercing a clean stream of bits from a bar-code front end required a significant effort using optics and analog electronics.

The vagaries of these disciplines have been brought in check as is evident in modern, digital-output bar-code wands. Bar-code wands operate directly from a 5-V logic power supply and output a digital representation of the symbol being scanned. To facilitate an interface to a variety of different decoders, the output stage often uses an open-collector driver.

There are a number of parameters that must be considered when specifying the optical characteristics of a bar-code wand. Luckily, industry standardization has limited the number of permutations. Briefly, the optical wavelength can be centered in the visible (red) or infrared spectrum. The advantage of using visible light is that if the bar-code label looks fine to you, it should appear likewise to the wand.

The other thing you must be concerned with is the optical aperture size. A small spot size responds accurately to bar edges, but also picks up small spots and voids. Conversely, if the spot size is larger than the smallest bars and spaces, then the wand will have difficulty resolving the pattern. An aperture size about 0.8x works well for most codes. Here again, standardization limits the choices between high resolution (6 mil) and low resolution (10 mil).

Listing I--The five basic steps involved in decoding Code-39 can be implemented in C.

```
#pragma large code

/* Constants */
#define StartCode '*' /* Code 39 start code */
#define StopCode '*' /* Code 39 stop code */
#define NoSample 0 /* Sample count end mark */
#define NoCode 0 /* No-translate return code */
#define NoDecode 0 /* No-decode return code */

/* Global variables */
unsigned int SampleData[512]; /* Raw sample count buff */
unsigned int *SamplePtr; /* Pntr into sample buff */
unsigned int SampleCount; /* Number of samples */
unsigned char DecodeData[33]; /* ASCII decode buffer */

/* External references */
extern unsigned char Decode39(void); /* Main decode routine */
extern unsigned char DecodeChar(void); /* Bar to ASCII decode */
extern void ReverseSamples(void); /* Sample buff reversal */

/* Code 39 decode routine */
unsigned char Decode39(void)

    unsigned char DecodeCount, DecodeByte;

    SampleCount = 0;
    while (SampleData[SampleCount] != NoSample)
        SampleCount++;

    if (SampleCount < 27)
        return NoDecode; /* Not enough samples */

    SamplePtr = &SampleData[0];
    if (DecodeChar() != StartCode) { /* Check start code */
        ReverseSamples(); /* Try reverse direction */
        SamplePtr = &SampleData[0];
        if (DecodeChar() != StartCode)
            return NoDecode; /* Can't find start code */
    }

/* Main decode loop */
    DecodeCount = 0;
    while ((*SamplePtr++ != NoSample) && ((DecodeByte =
        DecodeChar()) != NoCode)) {
        if (DecodeByte != StopCode) /* Store data character */
            DecodeData[DecodeCount++] = DecodeByte;
        else {
            DecodeData[DecodeCount] = 0;
            return DecodeCount-1; /* Stop code found */
        }

        return NoDecode; /* Unable to decode */

/* Generate ASCII character from bar/space pattern */
unsigned char DecodeChar(void)

    static code unsigned char BarTable[4][25] = {
        {0,0,0,'7',0,'4',0,0,'2',0,'9',0,'6',
         0,0,0,0,'1',0,'8',0,'5',0,0,0,'3'},
        {0,0,0,'G',0,'D',0,'J',0,0,'B',0,'I',0,'F',
         0,0,0,0,'A',0,'H',0,'E',0,0,0,'C'},
        {0,0,0,'Q',0,'N',0,'T',0,0,'L',0,'S',0,'P',
         0,0,0,0,'K',0,'R',0,'O',0,0,0,'M'},
        {0,0,0,'-',0,'X',0,'*',0,0,'V',0,'0',0,'Z',
         0,0,0,0,'V',0,'.',0,'Y',0,0,0,'W'}
    };
```

(continued)

Listing 1-continued

```
    unsigned int *TempPtr, Threshold;
    unsigned char Bars, Spaces, c;

/* Generate reference threshold */
TempPtr = SamplePtr;
Threshold = 0;
for (c = 0; c < 9; c++){
    if ((*TempPtr) == NoSample)
        return NoCode;
    Threshold += *TempPtr++;

Threshold /= 8;
Bars = 0;
Spaces = 0;

/* Build binary bar/space image */
for (c = 0; c < 4; c++){
    if (*SamplePtr++ > Threshold)
        Bars |= 1;
    Bars <<= 1;

    if (*SamplePtr++ > Threshold)
        Spaces |= 1;
    Spaces <<= 1;

if (*SamplePtr++ > Threshold)
    Bars |= 1;
Spaces >>= 1;
```

(continued)

SAMPLING

The first step to decoding a bar code is acquiring the bar-space data. More specifically, information describing the bar-space widths must be recorded. This sampling can be performed in a number of different ways and, as usual, the appropriate method depends on what else is expected of the system.

Dedicated implementations, in which the system can dedicate all processor resources to sampling, permit the use of a simple software loop for counting the bar-space durations. Alternatively, it may be desirable to give the processor assistance from a hardware timer in lieu of using a software-based timing loop. Both these cases rely on the premise that the system can somehow vector off to the sample loop before too much of the first bar is lost.

If it is possible that the system may be off performing other tasks when the bar-space data starts coming in, then obviously the processor must suspend these operations promptly or

► Good Stuff ◀

Bar Code Sensor
Battery Controllers
Clock/Calendars
Digital Power Drivers
DTMF & Phone Interfaces
Firmware Furnace Widgets
HCS-II Hard-to-find Parts
I²C Bus ICs
IRLEDs & Photodiodes
IR Data Link Parts
IR Remote Control
Laser Diode Controllers
Linear Hall Effect Sensor
Multiplexers & Crosspoints
Power Op Amp
Remote Temperature Sensor
Stepper Motor Drivers
Watchdogs & Power Monitors
8051 Information
and more!

Use a soldering iron? Get the parts!

UPS: Ground/2nd day \$6.50/9.00 to 48 US states, COD add 54.50. PO Boxes and Canadian addresses: \$6 for USPS mail. Check, MO, or COD only; no credit cards, no open POS. NC residents add 6% sales tax. Quantity discounts start at five pads. Data sheets included with all parts.

Call/write/FAX for seriously tempting catalog...

Pure Unobtainium

► Your unusual part5 source ◀

13109 Old Creedmoor Road Raleigh NC 27613-7421
FAX/voice (919) 676-4525



Cimetrics
TECHNOLOGY

*Linking Microcontrollers.
Breaking Boundaries.*

The 9-Bit Solution

The Cimetrics Technology Y-Bit Solution is a complete microcontroller network (µLAN) that supports the 8051, 68HC11, 80C186EB/EC, and many other popular processors. The 9-Bit Solution takes full advantage of microprocessor modes built in to microcontrollers. The 9-Bit Solution allows simple and inexpensive development of master/slave multidrop embedded controller networks.

- 8051, 68HC11, 80C186EB/EC compatible
- A full range of other processors supported
- Up to 250 nodes
- 16 Bit CRC error checking with sequence numbers
- Complete source code included

Link Your Product
With A Cimetrics
µLAN Today!
617.350.7550

55 Temple Place • Boston, MA 02111-1300
Ph 617.350.7550 • Fx 617.350.7552

the first sample will be hopelessly distorted. If this is the case, you can use an interrupt to simply yank the processor into a dedicated sample loop where it stays until the sampling phase completes.

If you've got to stay live while servicing other real-time events, then there's no choice but to sample completely under interrupt control. This technique mandates the use of a hardware timer that is stopped, read, and rearmed every time an interrupt event occurs. You must provide a means of generating an interrupt on each transition, and the interrupt should be given high priority. Also, most timer systems have the capability of interrupting on terminal count. This is exactly what you want to pull you out of sampling after you've entered the trailing quiet zone and data transitions have ceased.

Some systems may have to deal with real-time events that are more critical in nature than the incoming bar-code data. This situation can be handled provided your processor has a timer-capture system. In such a system, the sample count is copied into a capture register from a free-running timer without stopping the timer. This happens automatically under control of a hardware pin that can also be used to assert an interrupt when a transition event occurs. The processor has until the next event to read the captured count before it is overwritten, resulting in a sample loss.

Very accurate timing measurements can be achieved using such a system. Of course, the sample buffer requires some manipulation to adjust all samples to look like zero-referenced up counts. Also, setting the proper duration for the timer-overflow interrupt requires additional overhead. (For thoughts on general-purpose sampling techniques, take a look at my column in *INK 30*.)

For my sampling routine, I'm taking advantage of the simplicity of the dedicated software method, although you'd seldom be able to use such a primitive technique in a real-world application. Since I'm primarily interested in showing you how to decode bar code, I won't waste space

Listing 1-continued

```

/* Now do lookup based on bar-space combination */
if (Bars > 24)
    return NoCode;

switch (Spaces) {
case 0x4: return BarTable[0][Bars]; /* 0100b */
case 0x2: return BarTable[1][Bars]; /* 0010b */
case 0x1: return BarTable[2][Bars]; /* 0001b */
case 0x8: return BarTable[3][Bars]; /* 1000b */
case 0xe: { /* 1110b */
    if (Bars == 0)
        return '$';
    break;

case 0xd: { /* 1101b */
    if (Bars == 0)
        return '/';
    break;

case 0xb: /* 1011b */
    if (Bars == 0)
        return '+';
    break;

case 0x7: /* 0111b */
    if (Bars == 0)
        return '%';
    break;

default: return NoCode;

/* Do sample buffer reversal */
void ReverseSamples(void

    unsigned in *Ptr1, *Ptr2, Count, Temp;

    Count = SampleCount-1;
    Ptr1 = &SampleData[0];
    Ptr2 = &SampleData[Count];

    for (Count /= 2; Count != 0; Count--) {
        Temp = *Ptr2;
        *Ptr2-- = *Ptr1;
        *Ptr1++ = Temp;
    }

```

going into bar-code sampling in any detail. For information purposes, let me briefly describe the steps taken by my rudimentary software sample loop.

Coming from an idle state, control is transferred to the sample routine on detection of a data transition (the first bar). The routine now initializes some general variables and starts incrementing a counter register until the data line changes to the opposite polarity. Once this change occurs, the count is stored, the storage pointer incremented, and the procedure begins all

over again. This cycle continues until the counter reaches some terminal value (due to lack of transitions) at which point the trailing quiet zone is recognized and the routine terminates.

Since the count interval is referenced to the loop time, this parameter can be tuned to accommodate the range of values which are encountered. Assume a nominal x dimension of 0.020", a wide-to-narrow ratio of 3:1, and a 10x quiet zone. A realistic scan rate would typically fall in the range of 5-30" per second.

To accommodate these parameters, the sample counter is 16 bits wide. The sample loop time is set to about 2.5 μ s. Terminal count is reached after an interval of 10 ms, and in the absence of transitions, this is the overflow count. To save space for the decoding algorithm, the sample routine listing is not presented here. However, the BAR. Z I P archive is available on the Circuit Cellar BBS and contains this and related modules. (If you do decide to examine the sampling routine, remember that it is set up to run on a 25-MHz DS80C320.)

DECODING 39

In keeping with my goal of providing a simplified firmware presentation, I will demonstrate the essence of Code 39's decoding algorithm. This is in fact an implementation of the logic described in the *Automatic Identification Manufacturers (AIM) Reference Decode Algorithm* for USS-39.

At this point, it would be useful to make a couple of general observations. This decode algorithm presents the basic steps for deciphering a Code-39 symbol. The underlying logic is sound, but incomplete. As the AIM specification points out, you would undoubtedly want to add secondary checks for acceleration, intercharacter gap, and absolute dimensions for any serious application. You should also realize that these secondary checks and balances can generate as much code as the algorithm. As a result, the logic of the algorithm can become totally obscured.

The other relevant issue falls smack in the realm of advancing the state of the art. It's not unusual to encounter bar-code labels that don't meet specifications. This may be due to dimensional-tolerance problems, poor print-contrast ratio, inadequate quiet zone, and suchlike. If some clever programmer comes up with a superior algorithm which consistently reads deficient labels (one that doesn't result in an increase of missed decodes, of course), this has an unsettling effect on the status quo. These things happen and illustrate the fact that meeting the specification should be just a starting point.

Q • How do you know you're getting the most from your development tool purchase?

A • Compare Avocet Systems with the competition.

- A Broad Line of High-Quality Products at Competitive Prices
- Free On-Line Technical Support for Registered Users. No Voicemail!
- Attractive Multi-User Discount Prices & Our "50%+" Educational Discount Plan!
- Unconditional 30-Day Money-Back-Guarantee

Now call the obvious choice!

AVOCET
SYSTEMS; INC.

The Best Source for Quality Embedded System Tools

(800) 448-8500

(207) 236-9055 Fax (207) 236-6713

ANSI-C COMPILERS

8051 • 68xxx • 68HC11
6805 • Z80/180/64180
6801 • 6809
H8/300 & More

MACRO ASSEMBLERS

8051 • 68xxx • 68HC11 6805
• Z80/180 • Z8 • 8096/196
6800/6801 • 6809
H8/300 • 8048 & More

SIMULATORS/DEBUGGERS

8051 • 680xx • 68HC11 6805
• Z80/180 • 6800
6809 • 8048 • 6502
8085 & More

AND HARDWARE

EPROM Programming & Emulators by EETools, Trbal, Softaid, Cactus Logic

#129

TURBO-128 THE NEXT GENERATION EMBEDDED CONTROLLER

★ ★ NO DEVELOPMENT TOOLS REQUIRED ★ ★

READY TO PROGRAM IN BASIC OR ASSEMBLY

Photronics Research introduces the T-128:

A True Single Board BASIC Development System The T-128 is based on Dallas Semiconductor's new 8051-compatible DS80C320

With its 2X clock speed (25MHz) and 3X cycle efficiency, an instruction can execute in 160ns; a 8051 equivalent speed of 62.5MHz!!! Equally impressive is the T-128's high-speed NVRAM

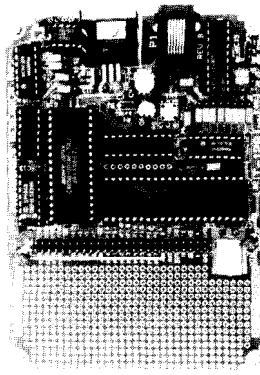
interface. Any of the 128K RAM may be programmed directly from a PC file through the console: eliminating EPROMs and associated tools. Program Development has "ever bee" faster or more convenient, eve" with the finest EPROM emulator. The T-128 features PORT 0 bias and EA-select for DS87C520 upgrade.

PROCESSOR

- Dallas Semiconductor's DS80C320
- 3W% more efficient than the 8051
- Three 16-bit Timer/Counters
- 13 Interrupts (6 Ext, 7 Int)
- A second 16-bit Data Pointer
- 384 Bytes of Internal RAM
- Programmable Watchdog
- Brownout Protection
- Power-Fail Reset/Interrupt
- Power-On Reset
- Fully supported by Franklin C51

MEMORY

- Entire 128K Memory Map populated with fast NVRAM (64K DATA + 64K CODE)
- All memory programmed on-board
- Partitionable as CODE/DATA/OVERLAD
- Code Space is Write-Protectable
- State-of-the-Art Data Protection



Only 5" x 3-3/4" • 500-hole Proto Area • Console/Power connected by a single 4-conductor telephone wire (very convenient)

BASIC520

- Modified BASIC-52 Interpreter (BASIC-520)
- Now Fast Enough for New Applications
- Stack BASIC Programs and Autorun
- CALL ASM Routines for Maximum Speed

I/O

- Three 8-bit Parallel Ports
- Two Full-Duplex RS232 Serial Ports
- Decoded Device I/O Strobes
- 50-Pin Bus Connector

UPGRADE

- OS87C520 processor (33MHz)
- Instruction cycle 121 ns
- 8.25 MIPS
- 8051 equivalent 82.5 MHz
- Internal 16K ROM/1K SRAM

Comes Ready to Run with power adapter/cable assembly. Includes utility diskette with DETAILED TECHNICAL MANUAL \$199 in OTV.

109 Camille St. • Amite, LA 70422 • (504) 748-9911 • Tech Support (504) 748-7090 • FAX (504) 748-4242

#130



Photo 1--Running the *test* code on a DS80C320 processor yields properly decoded data displayed on the LCD display.

The basic steps in decoding Code 39 are:

- 1) Confirm a leading quiet zone
- 2) For each character,
 - measure and assign total character width to S
 - compute threshold, $T = \frac{S}{8}$
 - build binary bit strings for bars and spaces
 - determine if the pattern matches a valid character
- 3) If the first character is not a start/stop code, reverse buffer and try again
- 4) Read until valid start/stop code is found [or until out of samples]
- 5) Perform secondary checks

These basic steps are implemented in the source code contained in Listing 1. This C implementation begins with the main decode function called `Decode39`. This function first counts the number of samples and determines if there are enough to continue. If the minimum number of samples is available, the `DecodeChar` function is invoked. This function actually does the work.

`DecodeChar` begins by summing the nine samples that (presumably) compose a character. A constant is applied to this sum resulting in the narrow- or wide-reference threshold. The code sequentially compares the

character's sample counts to this threshold and builds a binary representation of the bars and spaces. Using the binary-space pattern, a switch statement is performed. The first four cases handle the "normal" Code-39 characters and isolate ASCII code to the look-up table.

The table is in the form of a two-dimensional array that consists of 4 arrays of 25 elements each. Illegal codes are denoted by null codes. The four remaining "special" space patterns are directly validated and translated in the switch. The function now terminates and returns either a decoded ASCII code or an error code to the caller.

At this point in `Decode39`, the only valid character is a start code. If anything other than a start code is returned, the function assumes that this may have been a reverse scan and inverts the sample buffer. Following this, the pattern-matching procedure is performed again. If a start code is not recognized this time, the function terminates and indicates a no-decode to the caller.

If a start code is found, then the code falls through and indexes past the intercharacter gap and invokes `DecodeChar` again. If a displayable code is returned, it is placed into the `DecodeData` array. An invalid code causes the function to terminate

immediately and return indicating a no-decode. Detection of a stop code marks the completion of a good decode sequence. In this case, a trailing null is appended to the decoded data, and a value indicating the number of characters is returned to the caller.

DISCLAIM THIS

The functions I've presented all work individually and together. As evidence, Photo 1 shows the ec.32 SBC serving as the test bed in developing and testing the demonstration algorithms. The apparent performance of the system is actually quite good, and I encountered no problems with the system once I got the basic functions operational.

Where my discomfort lies is in the routines. I am well aware of the code's limitations, deficiencies, and omissions. That's not to say that I don't have a solid foundation on which to build, but clearly, the code is not finished.

From the user's perspective, this is not at all evident. At times like this, I wonder what lurks under the hood of some of the commercial software and systems. At least when I give you a weak algorithm, you get a disclaimer up front. ☹

John Dybowski is an engineer involved in the design and manufacture of embedded controllers and communications equipment with a special focus on portable and battery-operated instruments. He is also owner of Mid-Tech Computing Devices. John may be reached at (203) 684-2442 or at john.dybowski@circellar.com.

SOFTWARE

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

IRS

434 Very Useful
435 Moderately Useful
436 Not Useful

CONNECTIME

conducted by Ken Davidson

The Circuit Cellar BBS
300/1200/2400/9600/14.4k bps
24 hours/7 days a week
(203) 871-1988—Four incoming lines
Internet E-mail: sysop@circellar.com

With the start of our quarterly home automation inserts in this issue, I thought it only appropriate to spend this month's column dealing with home automation threads from the BBS. In the first discussion, we take a look at some of the potential pitfalls in trying to add intelligence to an HVAC system. While hot-water baseboard setups aren't particularly difficult to deal with, forced-air systems can be quite tricky.

In the other thread, we tackle a problem that comes up all the time in every on-line home automation forum I follow: flaky X-10 behavior. There is nothing cut and dry about power-line communications.

Fan control and HCS II

Msg#: 9252

From: DAVID WURMFELD To: KEN DAVIDSON

Is there a fan controller interface to the HSC II? I want to control the speed of my forced (hot/cold) air system. I am also looking for (digitally?) controlled air duct flapper valves. Eventually I would like to "shut down" the A/C in some rooms and not in others, so I would have to slow down the one service fan so as to not overpressure the reduced system. Any ideas for the "analog challenged"?

Msg#:11727

From: BILL NEUKRANZ To: DAVID WURMFELD

I'd be careful about trying to control furnace fan speed. Both your A/C and furnace units require good airflow to operate within safe limits.

You're correct to be concerned about pressure build up when running a zoned HVAC system. You should also be concerned about the liquid freon line getting too cold, eventually causing A/C compressor failure. And, during the heating season, you should be concerned about the furnace heat exchanger getting too hot. I'm operating a five-zone system for a 3400-square-foot home, with a single HVAC unit, and have protection for all three of these situations.

For the pressure, you can simply always run a "dump zone." That is, a zone that's always open in addition to any other zone. In my five-zone system, that would mean the minimum number of zones open would be two. Another

technique is to install a bypass duct that starts at the same point as all of your other ducts, and ends at the intake side of the furnace blower. In the middle of this duct you install a pressure valve. Adjust the valve such that it's closed when all zones are calling for air. I'm using the bypass valve solution.

The bypass duct also helps protect the A/C compressor by increasing air flow across the evaporator coil, keeping the liquid freon line from getting too cold. Additionally, I mounted a simple 45" temperature sensor switch directly onto the freon line. The switch is interfaced into the zoning controller. When the switch opens up at 45°, all air duct flapper valves open, maximizing air flow.

For winter heating, I've had to use a dump zone in past years. Otherwise, the furnace emergency-high-heat cutout switch would operate. I wasn't too interested in essentially "modulating" the furnace using this emergency protection. You'd have the same problem if you reduced fan speed without correspondingly reducing burner operation. This year, I have just finished installing a temperature sensor into the plenum distribution area that supplies all of the duct work. Once I figure out what is a safe temperature level, I'll program my controller to open up more zones when it gets too hot.

Some other things I'm doing that may give you some ideas for HVAC zoning:

1. I'm using balloons, not mechanical dampers, for what you're calling "flapper valves." They're very easy to install, especially for retrofit situations (like if your house is already built). I use an air pump to inflate or vacuum them. Much less expensive than the mechanical dampers. I got the equipment from Enerzone Co., in Dallas. All U.L. listed, too.

2. Get yourself a good controller if you're going to have three or more zones. I'm using an Enerstat five-zone controller. Works with heat pumps and forced air. Handles multiple stages of cooling (our A/C unit is a two speed unit). Also has digital inputs for unoccupied, high temperature limit, low temperature limit, and smoke alarm. I have all of these inputs connected to my home controller (not an HCS II, but performs similarly). The controller will make sure you don't overcycle the compressor, always have at least one zone open, shuts down and opens balloons in case

CONNECT TIME

fire alarm interface goes high, and so forth. Again, U.L. listed.

3. Put a PIR in each zone to sense room occupancy. Connect them to home controller and program it so the PIRs turn on and off the thermostats.

4. Install an analog temperature input from each zone to the home controller, too (separate from HVAC thermostats). Use this to program some maximum upper and lower limits when the PIRs have the zone thermostats turned off.

Msg#:13511

From: DAVID WURMFELD To: BILL NEUKRANZ

Thanks for the response, the balloons sound great. I have an 1800-square-foot ranch where all the heating ducts and heat exchanger is in the attic with easy access. I have a Century 2000 oil-fired forced-hot-air system with parasitic hot water. It is my intent to use the HCS II for control and other house chores. Would you be so kind as to post the address of the company that sells the balloons and inflators? Thanks again for the information.

Msg#:13767

From: BILL NEUKRANZ To: DAVID WURMFELD

The name and address of the company is

Enerzone Systems Corp
4103 Pecan Orchard La.
Parker, TX 75002
(214) 424-9808
Fax: (214) 424-8055

The fact that your furnace is in your attic makes the project easy, and Enerzone balloon dampers make retrofit of existing HVAC systems straightforward. You basically need a balloon damper and solenoid air switch for each zone, a pump, and a controller. For three or fewer zones, the solenoids and controller can be purchased as an integrated unit.

Ask Enerzone to send a catalog to you.

I'd be careful not to divide your home into too many zones without really paying attention to air volumes, pressure, noise (from higher air velocity), and furnace overheating (fire) safety. For 1800 square feet, I'd recommend no more than two or three zones.

Enerzone provides engineering services at no charge. Send them a sketch of your ductwork superimposed over your floor plan for recommendations. Include duct sizes and BTU rating of your HVAC system.

I'd strongly recommend that you not attempt to interface your HCS II directly to the furnace, or if you

decide to install zoning, interface directly to the air switch solenoids. Instead, interface your HCS II to a dedicated HVAC controller and let the controller handle all of the complexities needed for safe operation.

The HVAC controller I'm using provides the time delays needed for safe equipment operation, automatic heat-cool changeover (you need the same feature in your thermostats to take advantage of this), allows one thermostat to be set for heating and another for cooling (essentially "time slices" between furnace and A/C until all thermostats are satisfied), anti-short-cycle protection, high- and low-temperature alarm ports, smoke alarm port (shuts down HVAC and simultaneously inflates all balloons), and "unoccupied" port (ignore all thermostats).

Here are some ideas for what you can do with your HCS II in the world of HVAC. I'll illustrate with examples of how I integrated my home automation (dedicated processor made by HA1 and similar to an HCS II) and HVAC (another dedicated processor, made by Enerstat) systems.

(I have five zones. The five thermostats are wired into the HVAC controller. The controller outputs are connected to the furnace, air conditioner, and the five air switch solenoids. This basic setup will provide good energy savings and eliminate hot/cold spots in house.)

You can use programmable thermostats to increase energy savings. These work well if your schedule is always the same each day.

Like most people, though, my schedule is randomly different each day. So, I rely on a virtual "unoccupied thermostat" that takes control of the HVAC controller when I'm not home. I have a temperature sensor in the middle of the house, wired into an HA1 analog input. This is my "unoccupied" sensor. An HA1 output relay is connected to the HVAC controller's "unoccupied" port. Using the HA1 security subsystem's "Away" mode as a trigger that no one is home, I programmed the HA1 to disable the HVAC controller (via the "unoccupied" port) as long as the temperature sensor readings are within a programmable range. If the temperature falls outside of the range, then the HVAC controller is enabled, allowing the controller to use the five thermostats again.

Even when you're home, your movements throughout the house rarely mirror the temperature settings programmed into the five thermostats. So to maximize energy savings and convenience, I have PIR sensors in each room. These PIRs are connected to HA1 digital inputs. HA1 output relays switch in or out each thermostat in sync with PIR sensing. I have a 30-minute time delay set from the last motion sensed before a thermostat is switched out. To prevent a zone from getting too cold or hot, I have temperature sensors installed next to the thermostats. These

CONNECT TIME

sensors are connected to HA1 analog inputs. I programmed the HA1 controller to ignore a PIR and switch in a thermostat if temperature readings go outside of a programmed range.

For safety, I have an HAI-connected temperature sensor in the furnace plenum. HA1 output relays are connected to the HVAC controller's high- and low-limit ports. If plenum temperatures fall outside of a programmed range, the HVAC controller will sequence through a series of steps, starting with opening all balloon dampers, and ending with, if necessary, total shutdown.

I also have another HA1 output relay connected to the HVAC controller's smoke port. I programmed the HA1 to turn on this port if the HAI's fire subsystem goes into alarm (smoke/heat detector goes off or fire panic button pushed). The HVAC controller will respond by shutting down the furnace or A/C, and simultaneously inflating all balloon dampers.

Msg#:15771

From: DAVID WURMFELD To: BILL NEUKRANZ

Wow! Looks like I asked the right question at the right time. I'll take your advice and call the folks at Enerzone Systems. Thanks again.

X-10 troubleshooting

Msg#: 7612

From: DAVID CUNNINGHAM To: KEN DAVIDSON

I have been pulling my hair out for weeks trying to get a simple Radio Shack lighting circuit to not turn itself on. The load is two 150-watt incandescent floodlights and the controller is an RS timer. I have been using an identical circuit elsewhere in the building with never a problem, but on this particular light circuit, the light switch turns itself on usually around the same time each day.

I have tried every house code, and have swapped the two light switches. The problem is always in the same circuit. I should mention that the timer does turn the circuit on and off OK, but apparently something else is also turning it on. When I remove one of the floodlights, the problem seems to go away. But the other circuit that works OK has about 450 watts of incandescent lights on it, so I don't see that the troublesome circuit is overloaded.

I hate to bother you with such a mundane problem, but is there information available somewhere that would help

me troubleshoot this problem? Is there test equipment made that lets one monitor for X-10-type commands (or noise that would act like a command)? Thanks.

Msg#: 7617

From: KEN DAVIDSON To: DAVID CUNNINGHAM

There is often no explaining problems with X-10 setups. We've all had lamp and appliance modules that work fine one day, then mysteriously stop working the next. About all I can suggest is to make sure you have a signal bridge installed in your breaker box to ensure the signal makes it between the two hot sides. There is a signal strength meter available from Leviton (you can get it from most home automation places), but it's very expensive and not worthwhile for most homeowners.

One other option if you think noise is coming in from outside is to add a filter to the main power feed coming into your house. Such filters are available from most home automation suppliers, but you *must* have it installed by a licensed electrician. You can't simply flip off a breaker and work on a dead circuit to install it.

Msg#: 7817

From: DAVID CUNNINGHAM To: KEN DAVIDSON

I do have a signal bridge, but didn't bother to install it. When I started to put it in, I found that all the circuits I am using are already on the same transformer phase. But perhaps I'll try it anyway because whatever is inside the thing is apparently more than just a coupling capacitor. There are both black and white wires to connect.

This problem is really strange. The setup will work for a few days without any problem, then will turn itself on. I manually turn it off and a few minutes later it is back on again. It has never turned off by itself that I am aware of. Thanks for the help. I'll let you know if the bridge does any good.

Msg#: 7821

From: LEE STOLLER To: DAVID CUNNINGHAM

Sorry to butt in but.. .do you perhaps have a neighbor with an X-10 system that is using the same house code? That certainly could cause interference like what you describe..

Msg#: 7876

From: DAVID CUNNINGHAM To: LEE STOLLER

I don't know, Lee, but I have tried practically every house code and the problem persists. Also, I have a second identical lighting switch on another circuit in the same building which works fine. I have swapped switches so I know it isn't that. Oddly enough, the problem circuit will

CONNECT TIME

go for 1 to 3 days without acting up then turn itself on two or three times each day for a few days. I keep thinking there must be something wrong about the way the bad circuit is wired-like a reversed black and white wire, or maybe a ground fault. But I can't find anything. All this is happening in a small office building in which the wiring is in conduit either above the drop ceiling or below ground. So it isn't too easy to trace it out.

Msg#: 8045

From: LEE STOLLER To: DAVID CUNNINGHAM

Hmmmm.. .quite a mystery. You've eliminated the possibility that a human being, unknown to you, is coming into that office and turning the light on manually? Does that office contain supplies that someone else might want from time to time?

Msg#: 8055

From: DAVID CUNNINGHAM To: LEE STOLLER

Actually, if anyone wanted to break in here, I don't think they would bother with the office supplies or turn on the rear entrance floodlights. Today I noticed that the light turned itself on three times over a 45-minute period when I kept my eye on it (and turned it off manually whenever I saw it was on). I am increasingly convinced that there is some sort of spurious signal on the line that is causing the problem, and that it is either closer to or possibly in the circuit which the problem switch is in (because the front lights are never affected).

Do you know whether anything besides X-10 signals can cause an interference? I seem to recall, for example, that at one time there were intercoms that used the power line. Or perhaps a security system is using it. I don't think it is another X-10 signal because changing the house code does not help. But maybe some sort of broadband noise within the same signaling frequency range is doing it. Any ideas are greatly appreciated!

Msg#: 8366

From: LEE STOLLER To: DAVID CUNNINGHAM

Now the real can of worms opens.. .

There are zillions of RF generators out there! What you heard about intercoms is correct. Some *do* use the same sort of frequencies that the X-10 uses. There are also possibilities in other things. Have you tried turning off other devices in the building (except that troublesome light circuit) and seeing if the thing still goes on? You have to suspect everything. Fluorescent lights now use "energy saving" ballasts that actually are switching power supplies that can generate hash on the line. Computers use switching power supplies...ditto.

On another tack, what kind of controller are you using? Are you sure it's OK? Maybe it thinks the light is in security mode (due to some internal fault) and is turning the light on at random. Leave the light off and unplug the controller. See if the light still comes on.

Msg#: 8391

From: JOHN HARTMAN To: DAVID CUNNINGHAM

My upstairs neighbor used to have a PC clone which would turn on our X-IO dining room lights whenever he booted from floppy. Changing house codes did help somewhat, but the problem persisted until upstairs got a hard disk...

Dubious technology, X-10. I can't imagine running communications on power line without pretty hefty CRC validation, and I can't imagine that the PC upstairs generates the right CRC to turn the lights on. Doesn't give me a lot of confidence. On the other hand, my lights do what I want MOST of the time. :-)

Msg#:12909

From: CHRIS TYLKO To: DAVID CUNNINGHAM

I guess the two biggest problems with X-10 are:

- 1) Modules that do not turn on or off when they should, and
- 2) Modules that turn on when they shouldn't.

You're referring to the latter, in which I, unfortunately, have a lot of experience. The first thing you want to know is whether the module is going on because of a valid signal. There are cheap and not-so-cheap ways of determining this. Change the module address, or plug the same module in somewhere else. If you really want to know, use a TW523 with some inexpensive PC software such as that offered by Baran Harper. This will allow you to monitor all signals over whatever period of time you want and then save the data to disk.

If your problem is not a valid signal, then something down circuit from the module may be triggering it on, such as a loose connection. The fact that you're removing a light bulb and it works OK suggests the filament in the bulb may be damaged. As it vibrates it changes resistance enough so the module thinks you're flicking a switch and turning it on locally. Modules let a very low current through the circuit while it's in the off state so it can sense such a switch flicking (actually, it's not true of all models; some specifically don't work that way).

If that's not it, then you may be suffering from "poor quality power." In my terrible experience, the transformer feeding my house was faulty and saturated; the neutral was

CONNECT TIME

not pure, most probably due to moisture in the oil which causes small shorts. This is unnoticeable to even sophisticated line monitoring equipment. You see, for the X-10 system to work, it has to have a good neutral provided by the transformer.

If your problem is only limited to one module in one location, try checking to see that all connections from the outlet back to the main panel are secure. Also check (or have a certified electrician check) your main panel to make sure everything is snug and tight.

Finally, noise (outside of the circuit) has to be really bad to turn on a module; the X-10 binary address is VERY specific and virtually nothing on the grid looks like it.

Msg#:14905

From: DAVID CUNNINGHAM To: CHRIS TYLKO

I have been continuing the witch hunt for the cause of the erratic turn-on on one of my X-10 light switches based on some of the ideas you gave me. Thought you might be interested in the results.

[1] I am suspicious that the timer/controller itself may be responsible, because the light does not seem to go on on those days when I unplug the controller. This is not a definitive conclusion, because I have other things to do all day than watch a light out back to see if it goes on.

[2] I also think the ambient noise level in the X-10 frequency band is very large and is due to the switching power supplies used in the various PCs around the office. Here are some interesting results I measured using a Leviton X-10 coupler between a power strip and an oscilloscope. This coupler, from what I can determine, is some sort of tuned circuit which provides excellent isolation of the 60 Hz but couples frequencies around 100 kHz through. (I first tried using a pair of 0.1 μf caps, but they couple so much 60 Hz through that the 100-kHz stuff gets lost.) In all cases below, the unit under test was plugged into the same power strip as the coupler (and scope).

- Ambient level is about 5 mV peak
- '486 PC #1 measured 10 mV peak
- '486 PC #2 measured 40 mV peak
- '486 PC #3 measured 15 mV peak

[3] At the same time, I noticed some very large spikes at 120 Hz.

- Halogen light with dimmer measured 13 V peak (26 V P-P).
- Coffee pot (warmer) measured 10 V peak
- Laser printer measured 75 mV peak

Remember, all of these measurements were made through the X-10 coupler, so they represent the 100-kHz component of the actual noise or transient.

How does this compare with the timer's control voltage? If you plug the timer into the same power strip, it outputs about 1.5 V peak (3.0 V p-p). But if it is plugged in across the room into a separate circuit, it produces about 75–100-mV peak at the scope.

As you can see, the PCs contribute energy in the same frequency band as the controller that approaches that of the control signal in magnitude! What I would like to try now is to add some power line filtering to the PCs that would suppress this. Do you know if anyone makes such a thing that can be simply plugged in?

Msg#:19434

From: CHRIS TYLKO To: DAVID CUNNINGHAM

Very interesting! I tried to find my notes from two years ago when I went through the "unwanted lights on," but unfortunately came up empty.

OK, the easy part. Filters are available; as a matter of fact, Leviton has several different types which could probably help trap out the noise. Your readings indicate a lot of noise, and if I remember the (Leviton) X-10s were specified to work with up to 5 mV. I do recall being told, however, that noise on the lines could interfere with X-10 operation, but could not turn on a specific module.

This brings a source of help to mind. The X-10 people were of no help whatsoever; Leviton people, on the other hand, were extremely helpful. There was one fellow I spoke to at their tech line who was great (sorry but I can't find his name anywhere). If you live in the States, in a reasonably accessible area, and if you are using Leviton modules, they may even come by to help you out if your problem is "interesting enough." Unfortunately, I live in Montreal, and Leviton has no "X-10 qualified" people offering such help in Canada. There is a U.S. tech line for Decora Electronic Controls at 800-824-3005.

Msg#:22984

From: PELLERVO KASKINEN To: DAVID CUNNINGHAM

I do not have any X-10 equipment, so I may be off the mark, but here is my understanding of your situation and measurements.

The basic concept of X-10 communications is supposedly dependent on power line synchronization in such a way that the short bursts of signal take place at the zero crossing of the 60 Hz. This is a deliberate choice for both an easy implementation and because very few loads or controllers cause noise bursts at this exact time. I'll try to elaborate on this load-caused noise aspect.

CONNECTIME

You get high-frequency noise when a load is switched on or off. The worst noise generators are light dimmers and similar devices that control power level on every half cycle by phase control. In other words, the SCR or triac is turned on at any point during the sine-wave cycle. For a simple mental picture, let's assume the load is very pure resistor and we want about half the maximum power. So we set the trigger to the peak of the sine wave. The load before the trigger point sees zero supply voltage and within a micro-second or less, it sees 160 V. The transient represents anything up to about 1 MHz. Now, this is what the load experiences. What happens elsewhere in line depends on several small (and mandatory) details of noise filtering that the manufacturer of the dimmer has included and also from the impedances along the power line.

If we assume for simple calculations that the load resistance is 16 ohms, our transient can also be expressed as 10 A. This 10-A transient has to come from somewhere. In principle it comes all the way from the power plant, but in practical terms, the line impedance does not allow such high frequencies to travel the required multitude of miles. There are capacitances, often deliberate power-factor-correction capacitors, along the line. These inherent or intentional capacitances are the source for the transient currents. In fact, the dimmer itself contains some capacitors within the noise filter. But they are not sufficient to provide the whole 10 A, so some of it has to come from other capacitances along your power wiring.

As any wire has some resistance and most definitely some inductance, any traveling 10-A (or even smaller, attenuated by the noise filter in the dimmer) transient causes voltage spikes proportional to the residual current and the impedance in the line from the "ideal" supply point to any selected measuring point along the path. Just like your 13-V peak. The saving grace for the X-10 system is that this spike is (supposedly) outside the signal time window.

I was talking in terms of a single transient. Now, if we add the natural inductances in the load-the noise filter and the line-we get ringing at a reduced frequency. Pulling closer to the 100-kHz band of X-10.. It still is outside of the time window for X-10, which leads us to the question can this or some other similar control actually hit the zero-crossing point and penetrate (or at least overwhelm) the X-10 receiver?

If the load is more inductive, the necessary triggering points shift earlier. Or if we want the full output, we pull the trigger point earlier. It is conceivable that we hit it all the way back to the zero crossing, isn't it? Well, the controllers may not go quite that far and if they do, the resulting transient is much smaller, because at that point

no voltage exists so a O-A transient results. Again, pretty good for the X-10.

But wait a moment! We have only talked about the starting of the load current. It also ends on every power-line half cycle. Now, if we have just the right amount of inductance in an otherwise resistive load and/or in the filter and wiring, we can get a nice ringing where it really hurts. The transients at this point are small, but like you point out, so are the active signals.

Now, I don't claim this is *the* answer to your noise issue, but you might consider running a few tests. Get one of the commercial line filters, such as the Corcom VR series, available from most of electronics distributors and at least from Newark. Pick one with enough current rating for the highest load you suspect as a source of the noise and then wire your line to that appliance (or the dimmer) through the filter. Repeat your scope measurements and/or wait for the malfunctioning to happen or be eliminated. Move the filter to the next suspect, until you have the full picture. And please realize that the capacitances inside the filter affect the whole line impedance distribution so that the transients may not travel along the original path after you have connected the filter, maybe even idle somewhere. Just one of those small complications or "challenges." :-)

We invite you call the Circuit Cellar BBS and exchange messages and files with other Circuit Cellar readers. It is available 24 hours a day and may be reached at (203) 871-1988. Set your modem for 8 data bits, 1 stop bit, no parity, and 300, 1200, 2400, 9600, or 14.4k bps. For information on obtaining article software through the Internet, send E-mail to info@circellar.com.

Software for the articles in this and past issues of *The Computer Applications Journal* may be downloaded from the Circuit Cellar BBS free of charge. For those unable to download files, the software is also available on one 360 KB IBM PC-format disk for only \$12.

To order Software on Disk, send check or money order to: The Computer Applications Journal, Software On Disk, P.O. Box 772, Vernon, CT 06066, or use your VISA or Mastercard and call (203) 875-2199. Be sure to specify the issue number of each disk you order. Please add \$3 for shipping outside the U.S.

IRS

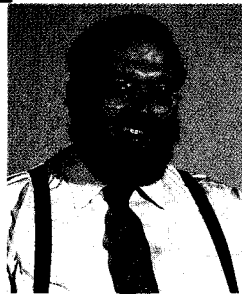
437 Very Useful

438 Moderately Useful

439 Not Useful

STEVE'S OWN INK

Hat Dance



O

hey say that the fun part of running a small business is that you never get bored doing the same thing. One day, you're making marketing decisions. The next you're fulfilling that wish list.

Around this place, it can be a real hat dance. In the same day, I might wear the hats of copublisher, editorial director, manufacturing, marketing, engineering manager, and line engineer I might decide on a range of issues from translating *Circuit Cellar INK* into Japanese, to what embedded control product to design for the next catalog, to allocating resources for a widget, to arguing that pin 15 is "chip select" and not "data out" after digging through a data book.

Now, this doesn't mean I get every hat. Surprisingly, in all these years, I have never played shipper or "faxer"—you know, I've never physically sent a fax. I did have to drive my plow in to dig the place out once last winter, however. I guess that's the snowman hat.

It's no secret that my favorite hat is still engineer. Of course, if you're one of the other engineers around here, sometimes my engineer hat gets considered the "impossible dream" hat.

For instance, after repeatedly coming across the same trade-journal ad for a popular new product that nobody else offers, I decided that such exclusivity was more than I could take. On went the engineering hat and less than two minutes later, Jeff and I were pouring over data books assessing the price-performance tradeoffs of making a superior product.

Unfortunately, the qualifications for wearing so many hats don't isolate you from competing interests. My marketing hat says, "Make it low cost and unique" while manufacturing pleads for reliability and ease of fabrication. Engineering says, "Cover all the bases or we'll have to do it again" while top management screams, "What the hell are you guys spending all this time and money on?"

If I'm not careful, I find myself being about as efficient as a committee. The only saving grace is that I ultimately tend to say, "Screw the cost. I want one to play with," and things actually get built.

This latest venture is driving me to take on a wizard hat. Despite a plethora of data books offering fabulous technology in Lilliputian packages, I am finding when I call for more details that today's latest science is an alchemical combination of vaporware and infinite lead times. It is taking true wizardry to mediate between the hats of purchasing and manufacturing to make changes in a design already in process, never mind the task of conjuring from this piece of gold something customers can afford.

Lamentably, I have no appropriate hat, short of one with a few tasteless bites chewed out of it already, for revealing to Jeff that the "committee" just changed the parts on the four-layer board he's been laying out for the last week. And oh yeah, Jeff, the whole thing still has to fit into less than a cubic inch!

A handwritten signature in cursive script that reads "Steve".