CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

#58 MAY 1995

COMMUNICATIONS

Exploring CAN (Controller Area Networks)

Tune in Radio-based Text

The Electronically Programmable Analog Circuit (EPAC)

PC Keyboard Emulation Continued

$3.95 U.S.
$4.95 Canada

EDITOR'S INK

Go Huskies!

While watching basketball this past weekend and pondering what I would write this month, I became amazed at how communications has shaped today's society. Here I was, watching the women's NCAA championship game half a continent away as it happened. My heart pounded out of my chest for the entire 40-minute game as our lady Huskies from the University of Connecticut battled in the toughest match of their undefeated season. In the end, they prevailed, topping off a phenomenal season. But we didn't have to wait for the evening news to find out. We were able to share in the glory of the moment as the victory celebrations took place on the court. Not quite like being there, but a close second.

(By the way, the men's team also had an outstanding season, making it to the NCAA "Great Eight." The campus is a mere 20 minutes from our editorial offices, so we got caught up in all the excitement.)

Not all such live communications might be considered positive, though. Take O.J. for instance (please, oh please). You can't turn on the television during the day now without bumping into trial coverage. Editorial omissions destroy any hope for impartiality. In this case, having a little less connectivity might make the whole fiasco more fair for all parties.

On a larger scale, last week I was exploring the World Wide Web with our new PPP Internet connection. I effortlessly connected with computers literally around the world from my home, each connection taking just seconds to establish. Granted there is a lot of trash on the net, but there is also so much neat stuff I could spend hours just reading and looking around. To think that I was using computers spread out all over the world was mind boggling.

Information superhighway, set-top boxes, cyberspace, multimedia, virtual reality.... Look past the hype of the popular press and there really are some exciting developments taking place. As engineers, perhaps we need to look beyond the technology we design to help ensure that it's used by society responsibly. For without that, those designs may do more harm than good.

Ken

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

EDITOR-IN-CHIEF
Ken Davidson

TECHNICAL EDITOR
Janice Marinelli

ENGINEERING STAFF
Jeff Bachiochi & Ed Nisley

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITOR
John Dybowski

NEW PRODUCTS EDITOR
Harv Weiner

ART DIRECTOR
Lisa Ferry

PRODUCTION STAFF
John Gorsky
James Soussounis

CONTRIBUTORS:
Jon Elson
Tim McDonough
Frank Kuechmann
Pellervo Kaskinen

PUBLISHER
Daniel Rodrigues

PUBLISHER'S ASSISTANT
Sue Hodge

CIRCULATION MANAGER
Rose Mansella

CIRCULATION ASSISTANT
Barbara Maleski

CIRCULATION CONSULTANT
Gregory Spitzfaden

BUSINESS MANAGER
Jeannette Walters

ADVERTISING COORDINATOR
Dan Gorsky

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) 6752751 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 696, Holmes, PA 19043-9613 or call (800) 269.6301.

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

Cover 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

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

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

Circuit Cellar BBS—24 Hrs. 300/1200/2400/9600/14.4k bps, 6 bits, 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 transfer 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 responsiblity for the safe and proper function of reader-assembled projects based upon or from plans. descriptions, or information published in *Circuit Cellar INK.*

INSIDE ISSUE 58

READER'S INK

HOW ABOUT MORE COMMENTED CODE?

Not good enough.

OK, OK. It's all my fault. I did not learn the C language yet, and I'm not going to do it any time soon.

I just don't like languages where expressions are typed on the keyboard's upper row (i.e., ones that use *\#^=|).

The undocumented piece of C code in Jeff Fisher's HP printer interface (INK 55) doesn't explain how he created the four-bit LRC.

Articles should be presented in human readable form. Do not assume that all readers know the language *you* use.

Remember, it is us, the dumb ones who need a helping hand. The smart guys don't need instructions. Think about it next time you're presenting an article—it's the dumb ones who will read it.

By the way, I just realized that I'm wrong again. I assumed that everyone's keyboard has the whole C vocabulary in the upper row.

Dusan Benko
Brooklyn, NY

For the benefit of our readers, let me hasten to point out that you made a third mistake. Circuit Cellar INK *only has* intelligent *readers.*

Now, to the real point of your letter. Yes, you're right. The code should have been commented. Unfortunately, Janice, who does the first pass through the articles, tends to assume that all engineers know more C than she does (a faulty assumption since she has taken an introductory course) and I, who know C, don't always think that I'm not the norm.

Profuse apologies. Tom Cantell would be the first to remind us right now of how assumptions make an "ASS-of-U-and-ME." We'll try to do better next time. For after all, if you had *the* comments, you could at *least write the code in your own favorite flavor.*

RETAINING HISTORICAL ACCURACY

Except in the life of a copy editor, I suppose this is a minor point. Take a look at *INK 54,* page *34.* It reads, "Developers of ARM were familiar with the 6502 from MOSTEK.. ."

While the 6502 was made by a number of companies, including Rockwell, it was developed by MOS Technology, which is not MOSTEK. MOS Technology was founded by a bunch of people who escaped from Motorola and who first created the 6501. The 6501 was pin compatible with the 6800 and 6502, differing only in having an internal clock and thus not requiring extra chips to create the clock. MOS Technology was later purchased by Commodore (or whatever it was called at that moment], which used the 6502 in the PET and Commodore 64.

Mike Firth
Dallas, TX

We apologize for the error and thank you for pointing *it* out to us. Hqwever, *I'd like* to remind you *that it is impossible to check the number of details that are in each issue. We pretty much have to rely on our authors to provide us with the correct details.*

I suspect in this case that the company MOS Technology had been verbally abbreviated to MOS Tech to such an extent that others began to assume the spelling of MOSTEK. Unfortunately, these sorts of fatal errors underly how languages both progress and regress.

Contacting Circuit Cellar

We at the *Circuit Cellar INK* 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, Circuit Cellar INK, 4 Park St., Vernon, CT 06066.

Phone: Direct all subscription inquiries to (800) 269-6301. Contact our editorial off ices at (203) 8752199.

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 (300–14.4k 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

PROXIMITY SENSOR

Senix has announced a distance-discriminating proximity sensor that features noncontact sensing with a push-button adjustment of the sensing distance. The **Ultra-100 Ultrasensor** provides distance detection and discrimination of objects from 6" to 10' with a repeatability of 0.1% of range and a resolution of 0.030". Applications include object-presence sensing, motion control, and level control.

Ultra- 100 is self-contained and housed in a flange-mounted, ABS plastic case. The sensor accepts 10–30-VDC power input and has a single-relay output with contact ratings of 2 A at 300 VDC or VAC. The relay-switching distance can be set to any point in the sensing range. Power and relay connections are accomplished through a 6', integral-pigtail cable.

A self-contained push button lets the user adjust the relay-switching distance, set the active relay state to normally open (NO) or closed [NC), and choose a switch response time of 50 ms, 500 ms, or 10 s. Push-button feedback and relay-state indication are provided by an integral LED indicator. A push-button lockout feature protects against accidental or intentional readjustment. A sensitivity adjustment accommodates target variations.

Ultra-100 lists for $199.

Senix Corp.
52 Maple St.
Bristol, VT 05443
(802) 453-5522
Fax: (802) 453-2549 **#500**

MEMORY CARD EVALUATION KIT

Advanced Micro Devices has introduced a **PCMCIA Flash-Memory Card Evaluation Kit,** which provides a complete and convenient tool for evaluating the AMD C-series flash memory cards. The kit permits users to evaluate the cards using multiple software drivers and flash file systems at a very low price.

Each kit contains an AMD l-MB C-series flash-memory card (AmC001 CFLKA), a user's manual, and the SCM SwapBox Lite PCMCIA card drive, which provides the physical connection between the flash memory card and the PC through a standard ISA bus.

Software provided with the kit includes four different versions of the SCM-FTL flash file system and Award CardWare 2.0, which offers socket and card services as well as driver user interfaces in DOS and Windows. Also included is Datalight's CardTrick FFS/ FTL flash file system, which provides both the FFS2 and FTL-compatible file systems as well as the capability to load both file system drivers simultaneously. SystemSoft's CardSoft+, which offers additional services and interfaces, also comes in the package.

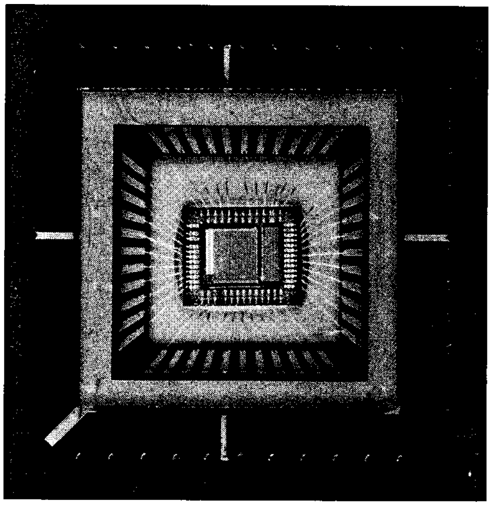
The kit requires a PC ISA slot and sells for $199.

Advanced Micro
Devices, Inc.
P.O. Box 3453
Sunnyvale, CA 94088-3453
(408) 749-5703
Fax: (408) 774-7216

#501

NEW PRODUCT NEWS

SINGLE-CHIP DIGITAL CAMERA



The Optical Systems Division of Marshall Electronics has just announced a digital video camera on a single integrated circuit. It is the first commercially available image sensor to have a built-in A/D converter, delivering a digitized black-and-white image through processor-compatible serial or parallel ports. The digital video-camera chip can be used for low-cost computer-video-imaging applications such as robotics, pattern recognition, highway traffic-flow monitoring, weather conditions, computer snapshots, and video telephones.

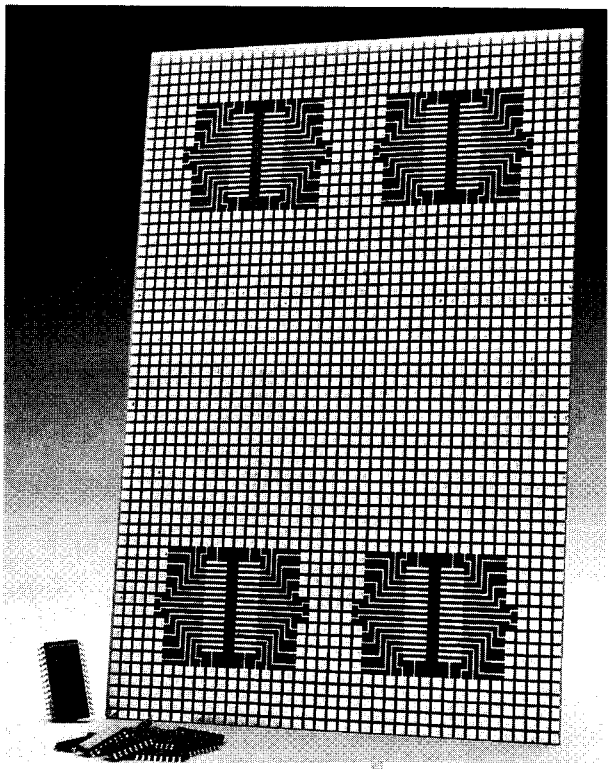
The integrated circuit uses proprietary CMOS sensor technology developed by VLSI Vision Ltd. The VVL-1070 has a 160 x 160 array of 10.5 x 10.5 µ pixels. Circuitry to drive and sense the array is packaged in a single Optical Quad Flatpack. The converter has an 8-bit digital output for serial or parallel interface. Features include analog output with sync pulses; wide-range electronic-exposure control for use with a variety of low-cost, fixed-aperture lenses; and automatic black-level circuitry. Power consumption is less than 100 mW.

An available Engineering Level Evaluation Kit reduces development costs and enables designers to rapidly develop a prototype using a defined interface circuitry. The kit includes a fully operational PCB which has an LCC with a glass lid mounted in an anodized aluminum enclosure. A C-mount, 12-mm lens and a wide-angle, 4.3-mm, fixed-focus lens are also included. An 18-way socket on the PCB brings out all major control signals. Solder pads make it easy to select various operating conditions.

The chip is available for $10 in quantity.

Optical Systems Division, Marshall Electronics, Inc.
P.O. Box 2027 • Culver City, CA 90231 • (310) 390-6608 • Fax: (310) 391-8926

#502



SURFACE-MOUNT PROTOTYPE BOARD

Surfboard is a surface-mount prototype board offering variable-size SOIC pads for flexible design and prototyping. The unique pad layout offers easy and reliable mounting of wide (400-mil) and narrow (300-mil) body SOIC styles while accommodating up to 32 pins. The 100-mil matrix of 80-mil squares is specifically designed to permit chip resistors, capacitors, inductors, SOT-style semiconductors, and DIP packages to be mounted. Surfboard brings the benefits of surface-mount technology to RF engineers and designers, offering more complex designs.

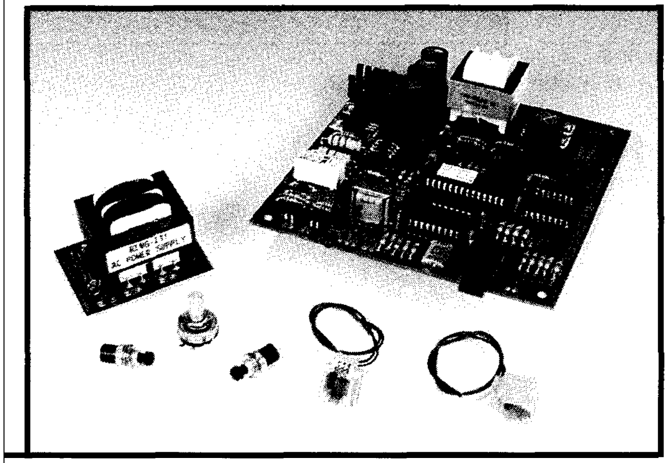
Surfboard measures 4″ x 6” and features 6 1-mil, FR4–fiberglass-epoxy laminate with solder plating over 1 -oz. copper on both sides of the board. A ground plane is included on the back side.

Surfboard sells for $19.95 (single quantity). Other sizes and designs are available.

ECode Systems, Inc.
7050 North Wilder Rd.
Phoenix, AZ 85021
(602) 870-8063
Fax: (602) 371-8736

#503

NEW PRODUCT NEWS



PHONE-LINE SIMULATOR KIT

Digital Product announces Ring-It!, a microprocessor-controlled telephone-line simulator which evaluates all types of standard telephone equipment. Ring-It! can be used to test or demonstrate any standard telephone, answering machine, fax unit, voice-mail system, or modem.

Telephone equipment connected to the test system behaves as if it were connected to a real telephone line. For example, a connected telephone produces an actual dial tone. Dialing a seven-digit number with a touch-tone phone rings a device plugged into the test line. Busy signals and reorder tones are also heard as with a standard phone line. An LED digital readout displays the DTMF digits that are dialed to verify touch-tone buttons.

Five different test modes offer standard telephone-line emulation or special repetitive-cycle testing. The LED readout shows the test mode in use. An internal 20-Hz ring generator uses digital precision to ensure that any kind of telephone equipment can be rung. The ring circuit uses a superimposed voltage source like the phone company's to maintain compatibility with all popular telephone products.

The complete Ring-It! kit sells for $145 and comes with a two-piece printed-circuit-board set, all electronic components, programmed microprocessor, transformers, and a technical manual. The basic kit sells for $53.95 and includes the circuit board, programmed microprocessor, and technical manual.

Digital Products Company
134 **Windstar** Cir.
Folsom, CA 95630
(916) 985-7219
Fax: (916) 985-8460

#504

IN-CIRCUIT EMULATOR

NICE-52 is a low-cost, high-performance in-circuit emulator for 80(C)31, 80(C)32, 8x(C)51, and 8x(C)52 microcontrollers. It comes complete with 64 KB code memory, 64 KB data memory, a 40-pin DIP header to connect to a target board, and a high-speed ISA-bus interface card to connect to any PC. It can run at up to 16 MHz in real time and operate without a target system for preliminary software debugging.

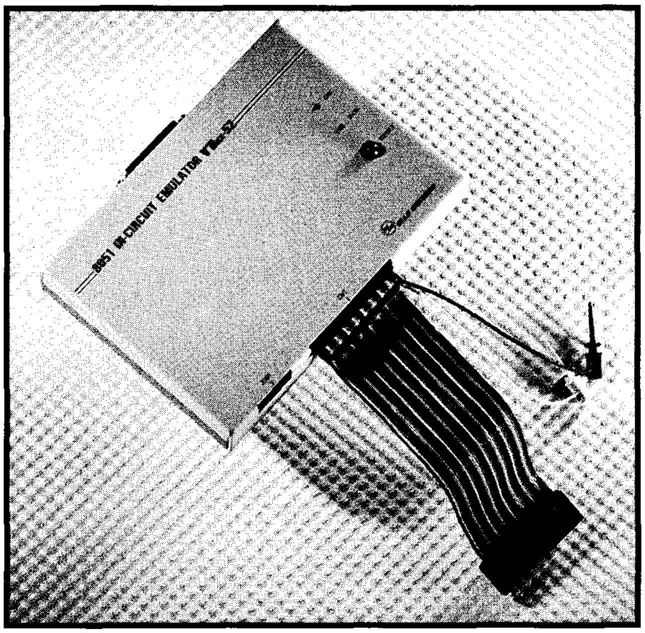
The software interface is easy to operate, yet powerful. Common operations are controlled by the keyboard function keys. Pull-down and pop-up windows offer editing and display functions. Any byte or bit address and all SFRs can be directly viewed and edited. An in-line assembler enables quick code changes.

The main screen displays all SFRs, internal and external data memory, and the program code as assembly or C instructions. A watch window enables viewing of up to seven C-level variables. Powerful instructions are available such as single step, go slow, go in real time until a breakpoint is reached, single step but execute subroutines in real time, and so on. Up to 10 breakpoints can be set to trigger on an address or address range in combination with fetch, read, and write conditions and an external signal-input clip. NICE-52 is compatible with most third-party assemblers and C compilers.

NICE-52 sells for $895.

Tribal Microsystems, Inc.
44388 S. Grimmer Blvd.
Fremont, CA 94538
(510) 623-8859
Fax: (510) 623-9925

#505



NEW PRODUCT NEWS

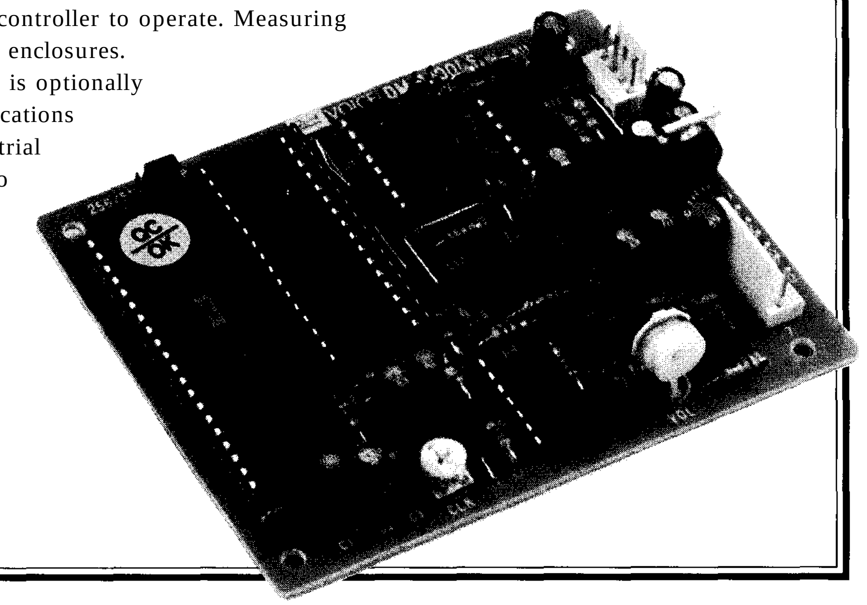
LOW-COST VOICE BOARD

The DMlOOOLS from Eletech provides high-quality audio at low cost. When activated by external contact closure or a motion sensor, it plays the message stored in its EPROM chip. Up to two minutes of message can be predigitized and programmed into EPROMs with a low-cost voice development tool.

The DMlOOOLS is powered by a single 6–12-VDC supply. The audio output is up to 2 W into a 4-Q speaker. Standby current is only 1 µA, thanks to onboard power-management circuitry. The board is totally self-contained and needs no controller to operate. Measuring only 3” x 3.5″, the board easily fits into tight enclosures.

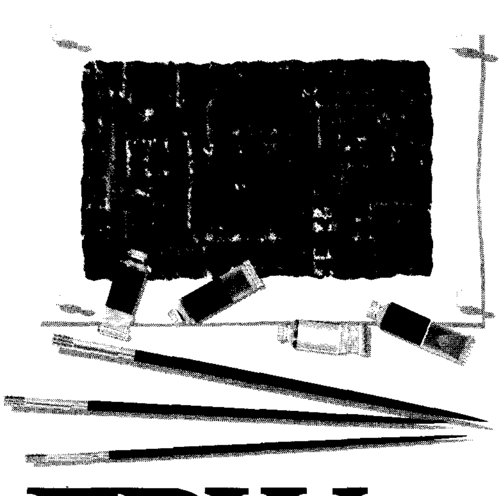
A passive infrared (PIR) motion detector is optionally available for use with the DMlOOOLS. Applications for the board include talking displays, industrial control, talking alarms, home control, and so on.

The DMlOOOLS (without EPROM) sells for $30 (single sample) and $20 (quantities of 100).



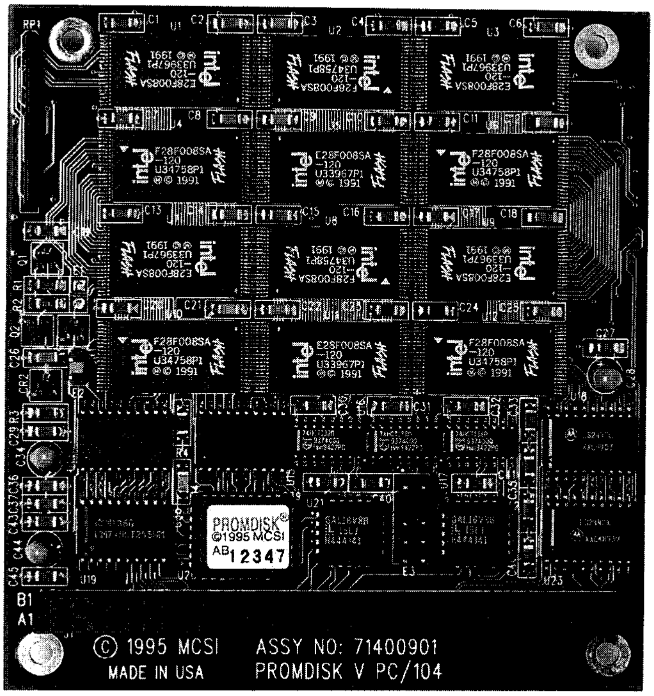
Eletech Electronics, Inc.
16019 Kaplan Ave. • Industry, CA 91744
(818) 333-6394 • Fax: (818) 333-6494

#506



NEW PRODUCT NEWS

SOLID-STATE DISK BOARD



A **32-MB** solid-state disk-emulator board has been introduced by MCSI. The **PROMdisk V** emulates up to two bootable read / write fixed or floppy disk drives with capacities of l-32 MB using onboard flash memory. The board includes an onboard BIOS extension ROM which contains the Datalight FTL integrated flash file system and boot utilities. It is fully DOS and Windows compatible, enabling the user to copy, read, and erase using standard DOS commands.

PROMdisk V replaces mechanical disk drives in systems designed to operate in harsh environments or where weight and size is at a premium. The use of the PROMdisk in embedded or dedicated applications offers substantial benefits in overall system cost, performance, and reliability. Since the board operates at bus speeds and is not encumbered by mechanical latency, the average read / write throughput is dramatically increased over that of a typical hard disk drive. Since there are no mechanical moving parts, PROMdisk V requires less power and operates cooler, resulting in a substantial increase in system reliability.

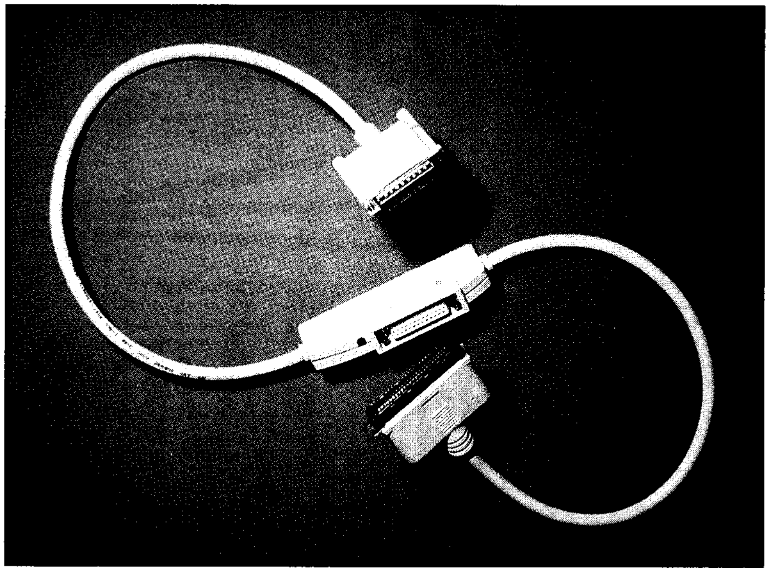
The PROMdisk Disk Emulator Board comes with the onboard FFS firmware, Datalight ROM-DOS, a user's manual, and utility diskette. The price of the PROMdisk V with 1 MB memory starts at $149.

Micro Computer
Specialists, Inc.
2598g Fortune Way
Vista, CA 92083
(619) 598-2177
Fax: (619) 598-2450

#507

PARALLEL TO SCSI ADAPTER

Shuttle Technology announces Shuttle Connection, a parallel-port-to-SCSI adapter for notebook, laptop, and desktop PC users. The connection lets users link and share all of their current SCSI peripherals without the need of host adapters. The unit meets the demand for low-cost, high-performance converters, providing parallel-port connectivity for all SCSI peripherals. It operates with DOS, Windows, OS/2, and NetWare.



Shuttle Connection operates with all parallel ports including the new IEEE 1284 enhanced parallel port at data transfer rates of up to 1.5 MBps. It includes installation software with each plug-and-play adapter connecting SCSI CD-ROM, hard disk, SyQuest, Bernoulli, tape, floptical, and magneto-optical drives via the parallel port. The unit is powered from the SCSI device or optional 5-V supply.

Shuttle Connection sells for $169.

Shuttle Technology, Inc.
38069 Martha Ave., Ste. 400
Fremont, CA 94536
(510) 505-0567 • Fax: (510) 505-0581

#508

FEATURE ARTICLE

The Solution's in the CAN—Part 1

Industrial-class, Small-area Networks

Brad Hunting

In the first of a two-part series on CAN, Brad looks specifically at the CAN protocol and how it and its existing hardware implementations meet the requirements of a small area network.

As our industry becomes increasingly complex, we often find it difficult to respond to the demands placed on us. Bigger projects come along and to conquer them, we learn to work in teams and with outside support. Tasks get delegated and we learn to rely on the expertise of others.

Essentially, the trend toward larger scale forces us to move from a centralized scheme, where we individually control and produce all aspects of the project, to a distributed scheme, where we work with peers and tackle large problems together.

In the same way that large tasks are broken down and solved by distributed teams, large centrally controlled systems are broken down and solved by distributed control systems. The shift from centralized to distributed control characterizes technological changes. For instance, note the move from centralized mainframes to distributed workstations.

Distributed control represents many advantages:

- fault tolerance-if one unit fails, the whole system does not necessarily fail.
- ease of development-if the protocol linking the distributed controllers is

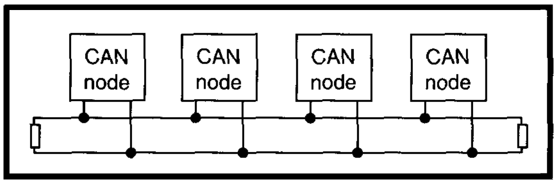


Figure 1—*CAN's multidrop bus topology allows easy-on, easy-off bus access.*

well defined, the implementation of one node does not affect the implementation of other nodes.

- simplified wiring and interfacing—the control processing occurs at the point of action. A controller can be placed at the sensor or actuator location or integrated as the controller for the sensor/actuator system.

The backbone of any distributed system is a communication network which links the pieces together into a consistent whole. The ability to transfer information cleanly and efficiently is key to the success of a distributed project.

However, the decision to install a distributed control system requires you to decide which network to use. If you want a small-area network, there are many available choices including Arcnet, Bitbus, CAN, 9-bit serial, and so on. As you can imagine, each solution offers varying levels of support and success.

I have worked with a variety of small-area networks. My most recent favorite is the Controller Area Network (CAN) protocol developed by Bosch [1]. In this article, I'll discuss the CAN protocol and some of the hardware implementations. 1'11 look specifically at how the CAN protocol and its existing hardware implementations satisfy the requirements of a small-area network. Next month, 1'11 present hardware and software designs to implement a multinode PC and microcontroller CAN network.

However, let's start out with an overview of the CAN protocol. After mentioning some of the supporting features of the hardware implementations, I'll delve into the nitty gritty of how CAN works.

OVERVIEW OF CAN

CAN has been called a multicast, broadcast, or peer-to-peer network. Figure 1 illustrates how each CAN node interfaces to a linear bus. CAN is implemented as a half-duplex connection-one node transmits and all nodes receive. As well, any node can initiate a transmission at any time. This stands in contrast to a star, token-passing, or master-slave configuration.

CAN supports a data-centric model where the data, not its origin, is significant. Unlike other networks, CAN does not have the concept of node addresses, and no node address is associated with any message.

Instead, messages have identifiers and priorities. Any node can produce or consume information. Every node on a CAN bus transmits or receives any message. Certain messages are likely produced by certain nodes, and other nodes are likely to consume those messages-the source or destination of a message is irrelevant.

11 bits	Identifier
1 bit	Remote Transmission Request
4 bits	Data Length Code
0-8 bytes	Data

Figure 2-A *CAN data packet includes a unique 1 l-bit message identifier and C-8 data bytes.*

From a user point of view, a CAN message looks like Figure 2. Every message has a unique identifier, which also functions as the message priority. A bit differentiates a data packet from a Remote Transmission Request (RTR) packet. This differentiating bit is followed by a count of the number of data bytes and O-8 bytes of data.

Since every node receives every packet, CAN controllers implement an acceptance mask (see Figure 3). The acceptance mask screens unwanted messages, preventing the CAN controller from generating an interrupt to the host processor on every message.

CAN's message arbitration scheme requires nodes to monitor the bus while transmitting. To transmit a message that could also be received by the transmitting node (i.e., the message identifier passes the transmitting node's acceptance mask], the transmitting node must have a free receive buffer.

8-bit acceptance code register
8-bit acceptance mask register

Message is accepted if bitwise
[(8 MSB of ID) EQUAL (AC)] OR (AM) = $1111\ 1111_b$

Figure 3-CAN *controllers provide* message filters to *screen unwanted messages.*

A free receive buffer is required because, when a node initiates transmission, it does not know if its transmission will be preempted by a higher-priority message. CAN hardware implementations have multiple receive buffers (a minimum of two) to allow a message to be transmitted before a received message is read from the buffer.

CAN specifies a maximum data rate of 1 Mbps, but it does not specify a maximum number of nodes. The maximum realized data rate and maximum number of nodes is determined by the capabilities of the physical layer. Message overhead is 30–70%, depending on the number of data bytes sent with each packet. Protocol violation and CRC error detection are built into the CAN controller hardware.

POPPING OPEN THE CAN

It is important to note the CAN protocol does not specify the physical layer (i.e., the electrical connections or signal levels required to transmit information from one node to another). CAN only requires that the physical layer have the ability to generate recessive and dominant bits.

A dominant bit is able to overwrite a recessive bit when both are transmitted simultaneously by different nodes. A common implementation of a recessive-dominant bit scheme is an open-collector wired-OR (see Figure 4). Any node transmitting a recessive bit leaves the bus pulled up by a resistor, and any node transmitting a dominant bit pulls the bus to ground, thereby overriding any recessive nodes.

CAN 2.0

In 1991, the CAN specification was upgraded to Version 2.0 [1, 3]. This revision extended the message format to increase the identifier from 11 to 29 bits. CAN 2.0 is fully backward compatible with CAN 1.2. Messages from both versions are allowed on the same net. CAN 2.0 specifies Version 1.2 frames, identifying them as standard format frames. Version 2.0 frames, on the other hand, are identified as extended format frames.

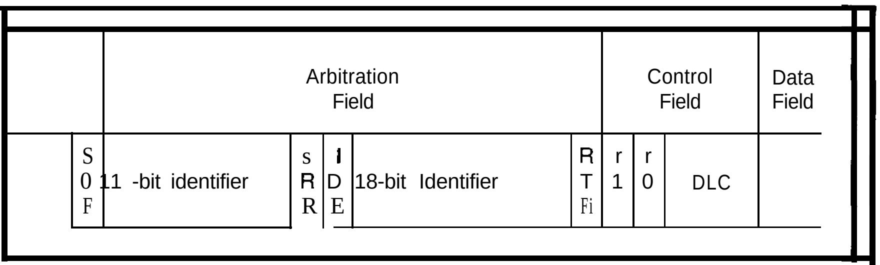


Figure I-CAN *V. 2 supports messages with a 29-M* identifier and is backward compatible with CAN *V. 1.*

A CAN 2.0 identifier and control field has the structure shown in Figure I. To distinguish between the Version 1 .0 and 1.2 standard format messages and Version 2.0 extended format messages, the reserved bit rl of CAN 1.2 is now noted as the IDE bit (see Figure II).

- Identifier (ID)-The standard format identifier length is 11 bits and corresponds to the base ID in extended format. The seven most-significant bits must not be all recessive. The extended-format identifier consists of 29 bits, a base ID with 11 bits, and an extended ID with 18 bits.
- Remote Transmission Request (RTR) bit-In data frames the RTR bit must be dominate. In a remote frame, the RTR bit must be recessive.
- Substitute Remote Request (SRR) bit-The SRR bit is a recessive bit which is transmitted in an extended frame at the position of the RTR bit in the standard frame. Collisions in which the base ID of a standard and extended frame are the same result in the standard frame winning arbitration over the extended frame.
- Identifier Extension (IDE) bit-The IDE bit belongs to the arbitration field for the extended format and the control field for the standard format. The IDE bit is transmitted dominate in the standard format and recessive in the extended format.

All other fields are the same in standard and extended formats.

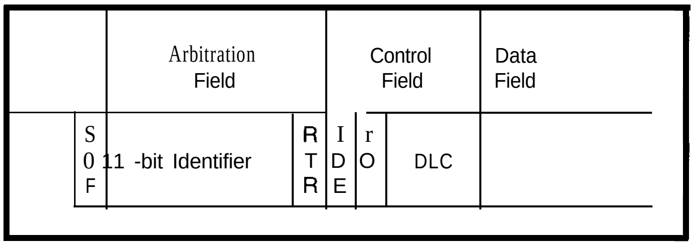


Figure II-CAN *V. 1 sports* an 1 l-bit identifier providing for 2032 unique messages on the net.

There are different methods of implementing the dominant-recessive configuration in hardware. This configuration is required to implement the CAN message priority arbitration. For short lengths and low bit rates, an open-collector wired-OR suffices.

WHAT CAN IS MADE OF

The CAN protocol specifies a packet-oriented protocol with the following four packet types:

- Data packets-these packets are generated by information producers and carry information to consumers.
- RTR packets-these request data from users and are generated by information consumers.
- Error packets-any node can generate an error packet to indicate a transmission fault.
- Overload packets-these packets request additional delay between data or RTR packets.

DATA PACKETS

Figure 5 shows the layout of a data packet, which consists of seven fields. The fields are start of frame, arbitration, control, data, CRC, ACK, and end of frame.

A start-of-frame field consists of a single dominant bit. A bus-idle condition is all nodes off the bus while the bus is in a recessive state. Any node can initiate a transmission when it detects an idle bus. All nodes synchronize to the start-of-frame bit of the node initiating the transmission.

The first field after the start bit is the 1 l-bit message-identifier field. The arbitration field contains the message identifier and acts as the priority of the message. If multiple nodes begin transmitting simultaneously, the first node to transmit a dominant bit overrides all nodes transmitting recessive bits (see Figure 4).

CAN requires transmitting nodes to monitor the bus. As soon as a node recognizes it has lost arbitration by sensing a dominant bit while it is transmitting a recessive bit, it immediately stops its own transmission.

A zero is considered .a dominant bit and a one a recessive bit. A lower-numbered identifier, one with more zeros in the front, presents a dominant bit earlier in the identifier, thereby overriding other nodes with lower priority messages. The lower-priority, higher-numbered messages lose arbitration.

This method of arbitration guarantees that neither time nor data is lost. This stands in contrast to Ethernet where a collision destroys data, causing all colliding nodes to initiate a random-timed back off. With CAN, the higher-priority node eventually overrides all lower-priority nodes and completes transmission.

Since CAN does not allow packet identifiers O-15, 11 bits generates 2032 distinct packet identifiers. The 1 l-bit

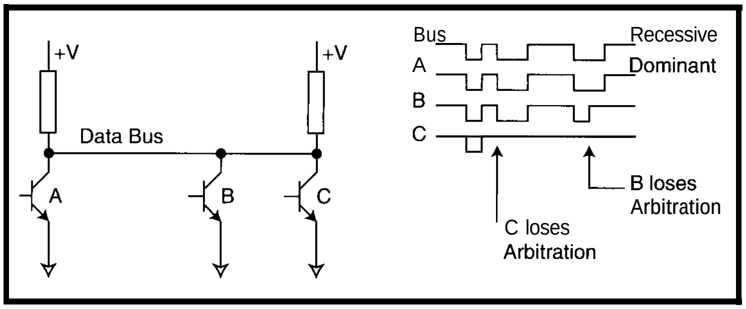


Figure 4—*CAN's dominant-recessive bitwise arbitration guarantees no lost time or data in the event of a bus-access collision.*

identifier has been changed to 29 bits in CAN 2.0 (see the CAN 2.0 sidebar).

The RTR bit follows the identifier. Since messages carry only a message identifier and not a node address, when a consumer needs information produced by a different node, it cannot request information from that specific node. Instead, in keeping with the data-centric model, the requesting node broadcasts a message using the message identifier of the information the node wishes to receive.

The RTR bit is set to a recessive state to indicate that this request is for a remote transmission of the data. Since all nodes on the net receive all messages, any node that can satisfy the remote request initiates a transmission of the data. This method enables the information to come from any node capable of producing the data.

Transmitting the RTR bit as recessive by the requesting node and dominant by the producing node prepares for the case in which the producer transmits the desired message at the same time as the requester transmits the request for the message. As soon as the producer transmits the dominant RTR bit, the requester recognizes lost arbitration and stops transmitting. The requester then immediately begins receiving the desired information.

In an RTR packet, the data-length count must be correct, but any data bytes are ignored by the transmitting circuitry. The control field contains two reserved bits (one of which has already been used for the definition of extended CAN), which must be transmitted as dominant.

The reserved bits are followed by the four bits of the Data Length Code (DLC). CAN only allows messages with O-8 data bytes, and these bytes must immediately follow the DLC. Following the data field is the 16-bit CRC field.

The ACK field is two bits long and contains a dominant bit followed by a recessive bit. The transmitting node sends both bits as recessive and expects at least one receiving node to send a dominant bit in the first ACK bit slot. If none of the nodes acknowledge the message, it is flagged as a transmission error by the transmitting node.

REMOTE TRANSMISSION REQUEST PACKETS

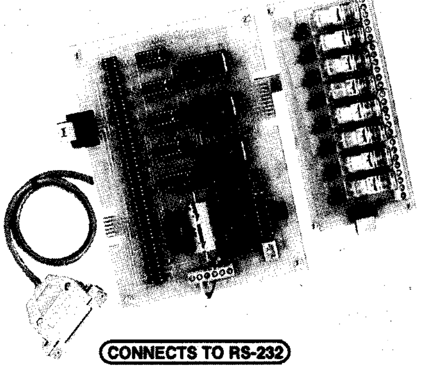
An RTR packet is identical to a data packet, except that the RTR bit is transmitted as recessive and the number of data bytes is always zero. An RTR packet DLC indicates the proper length of the data fields, but no data is transmitted.

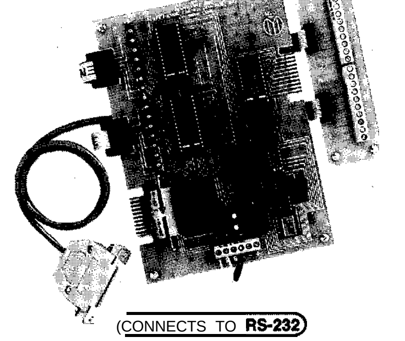
WHO'S USING CAN

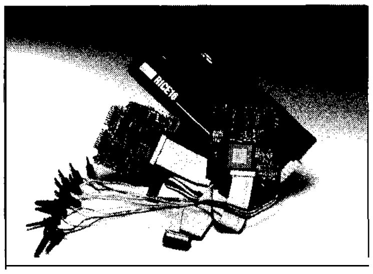
The Society of Automotive Engineers (SAE) endorsed CAN as a substantial portion of the data-link layer specified in *SAE Recommended Practices J1 939* [4]. Full use of *J1* 939 capability by U.S. automakers is expected in 1995 or 1996.

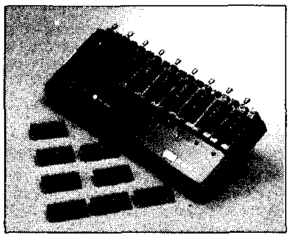
CAN has been widely accepted in Europe. European automakers began installing CAN in some production models as early as 1991.

Allen-Bradley's new DeviceNet and Honeywell's new Smart Distributed System (SDS) both use CAN for the hardware and low-level protocol layer.

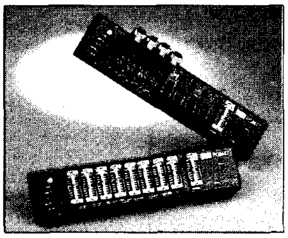








PGM16G: for 16C5x family



PGM47: for 16C71/84

PGM17G: for 17C42

ERROR PACKETS

CAN uses a bit-stuffing rule so that no more than five bits of the same state are transmitted consecutively. If five bits of the same state are transmitted, an additional bit is inserted with the opposite state.

Error packets are six consecutive dominant bits. Six consecutive bits of the same state violate the bit-stuffing rule, and all nodes register a message error. Allowing a single node to abort a message for the entire net ensures that data is consistent throughout the net.

If any node receives an erroneous packet, it blocks transmission and forces a retransmission of the packet. Error packets are initiated immediately on detection of a bit error and following the ACK delimiter if a CRC error occurs.

OVERLOAD PACKETS

Overload packets are similar to error packets but only occur during the interframe space between packets. Overload packets delay further bus activity for a short time. They also cause all nodes on the net to transmit seven recessive bits.

CAN specifies bit stuffing for the start of frame, arbitration field, control field, data field, and CRC fields. Error and overload packets are not bit stuffed.

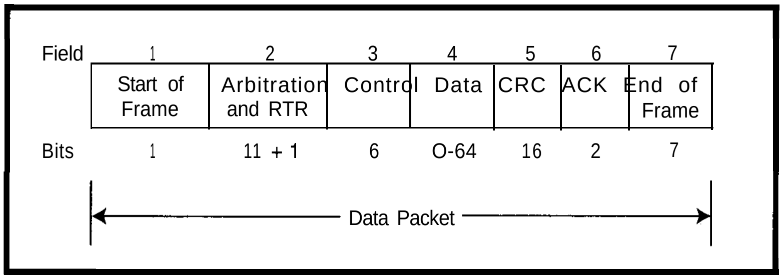
HOW DOES CAN STACK UP?

• Determinicity

The CAN protocol enables messages to be initiated any time the network bus is idle. Once a message is started, all of the bits of the message are clocked at a consistent rate. The bit rate is programmable over a wide range, but must be uniform for all nodes in the net. Given the bit rate and the priority of messages on the net, it is possible to determine the latency from the initiation of a message to the receipt.

. Medium to high speed

CAN specifies bit rates as high as 1 Mbps. The bit rate is fully programmable and must be appropriate to the transmission medium and noise environment.



Field	1	2	3	4	5	6	7
	Start of Frame	Arbitration and RTR	Control	Data	CRC	ACK	End of Frame
Bits	1	11 + 1	6	O-64	16	2	7

Figure *CA CAN message incurs 30–70% overhead depending on the number of data bytes and includes a CRC to ensure data integrity.*

- Error detection and fault tolerance

CAN specifies bit, stuff, form, acknowledgment, and CRC errors. Bit errors are detected when the transmitting node monitors a different state on the bus than what is being transmitted. Stuff errors occur when more than five consecutive bits of the same state are detected. Form errors occur when a fixed-form bit field contains one or more illegal bits. An acknowledgment error is detected by a transmitting node whenever it does not detect a dominant bit during an ACK slot.

Since CAN is data oriented rather than node oriented, nodes can enter and exit the net without reconfiguration or undue disruption of the net.

- Fault containment

CAN specifies a node to be in one of three possible error states: error active, error passive, and bus off. Error active is the normal state of a properly functioning node. An error-active node is able to transmit the six consecutive dominant bits required to initiate a message failure for all nodes.

An error-passive node transmits six recessive bits for an error. Otherwise, it transmits and receives data packets normally.

A bus-off node does not transmit any bits (either error or data) on the bus.

A CAN controller maintains internal counters for transmit and receive errors. Any time a good packet is transmitted or received, the appropriate counter is decremented by one to a minimum of zero. Any time an error packet is transmitted or received, the appropriate counter is incremented by eight.

When the error count reaches an appropriate threshold, the CAN controller switches to error passive and then a bus-off state. The nonproportional increment-decrement scheme ensures that a node experiencing repeated errors reaches an error-passive or bus-off state without causing excessive bus contention.

- Wire-short or open-fault tolerance

CAN controllers contains dual transmit and receive connections. CAN nodes function properly on a two-wire-plus-common or a single-wire-plus-common bus. This capability enables a net using two-wire-plus-common to reconfigure itself in the event of a single-wire failure—short or open.

The CAN output-controller registers define the function of the dual receivers and transmitters. Output configuration can be modified on the fly to compensate for faulty wiring. Detection of and compensation for bus failures is nontrivial [2].

- Physical layer

CAN is flexible enough to offer a variety of physical-layer implementations. As long as the physical layer supports the concept of dominant and recessive states, it can be used with CAN. Try any of these methods:

- DC coupled
- AC coupled
- transformer coupled
- direct one-wire or two-wire differential

The transmission medium can be one wire plus common, two wire plus common, or possibly fiber optic.

Maximum distance between nodes is determined by bit rate, location of sample point within the bit time,

AVAILABILITY OF CAN AND SUPPORTING TOOLS

Implementations of CAN-compatible controllers are being manufactured by Signetics, Intel, Motorola, NEC, Hitachi, and National. Tools for developing CAN applications are also available from these manufacturers and third parties.

Signetics produces the PCA80C200—a stand-alone CAN controller-and the 80C592—an enhanced 80C51-compatible microcontroller with a CAN controller built in. The Signetics controllers implement Basic CAN 1 .*x* protocol. The 80C200 is interface compatible with Intel and Motorola microprocessors.

Intel produces the 82526 and 82527 stand-alone CAN controllers. The 82526 implements full CAN 1 .*x* and the 82527 implements full CAN 2.0.

Motorola has introduced the MC68HC05*x*4 and ‘xl6 microcontrollers containing a built-in CAN controller.

- Small footprint-SOIC packages are available as well as a high-performance 80C3 1 and 68HC05 variants with built-in CAN controllers.
- Low power-Stand-alone CAN controllers can be placed into sleep mode to conserve power.
- Industrial temperature spec-Implementations of CAN are targeted at the automotive industry. Signetics’s and Intel’s controllers are specified for -40 to +125°C.
- Support tools-Stand-alone evaluation boards are available from individual manufacturers. Tools are available for net simulation, bus monitoring and analysis, and physical-layer implementation. CAN boards now appear for less than $200.
- Third-party support-Board-level solutions are also available from companies such as DIP. Philips provides a CAN prototyping board based on the 80C31.

frequency and tolerance of the controller oscillators, propagation velocity of medium, output delay of transmitter, and input delay of receiver.

CAN controllers have control registers which enable the bit sampling time to be set to compensate for media propagation delays.

CONCLUSIONS

CAN satisfies all of the requirements of an industrial-strength, small-area network. CAN has high throughput, capable error detection, a well-defined protocol, growing industrial support, implementation flexibility, multiple sourcing, and extended-temperature packaging.

As CAN expands in the industrial automation market, we can expect to see more support tools developed. If your needs include a robust, industrial-strength, small-area network, CAN offers a lot of benefits. ❑

I *would like to thank Roger McBride for his invaluable input and insight on this project.*

Brad Hunting has industrial experience in embedded systems development for ***process control. He has recently returned to school to complete a graduate degree at Rensselaer Polytechnic Institute. He may be reached at*** huntib@cat.rpi.edu.

REFERENCES

[1] Robert Bosch GmbH, ***CAN Specification V. 2.0*** (Philips Semiconductor, 1991) 1-68.

[2] Jens-Ulf Pehrs, "CAN Bus-Failure Management Using the P8xC592 Microcontroller," ***Philips Semiconductor App Note,*** HKI/AN 91 020, (1991) 1-16.

[3] C.P. Szydlowski, ***CAN Specification 2.0: Protocol and Implementations*** (Warrendale, PA: SAE Special Publications 921603, 1993) 2937.

[4] M.R. Stepper, ***Data Link*** Overview for ***Heavy-Duty Vehicle Applications*** (Warrendale, PA: SAE Transactions 902215, 1993) 723-737.

[5] James Pinto, "Two New Networks Target Factory-floor Devices," *I&CS* June 1994: 69-73.

[6] Charles Murray, "Dawn of the Smart Sensor," ***Design News*** May 9, 1994: 73-76.

[7] Michael Babb, "New Sensors Have Intelligence, Will Communicate," ***Control Engineering*** Feb 1994: 84-85.

[8] Mechatronics Editorial, "New Bus Links Devices on the Factory Floor," Machine Design Mar 21, 1994: 26, 27, and 37.

SOURCE

DIP, Inc.
P.O. Box 9550
Moreno Valley, CA 92552-9550
(909) 924-1730
Fax: (909) 924-3359

I R S

401 Very Useful
402 Moderately Useful
403 Not Useful

The RDS Prospector

Read Your Radio!

RDS enables a radio to display messages recovered from the inaudible 57-kHz band of the FM signal. Chris introduces us to an inexpensive decoder which displays text and works with an existing FM tuner.

FEATURE ARTICLE

Christopher Morris

You may not have noticed it, but a quiet revolution is taking place in North America's radio broadcasting. In 1993, the National Association of Broadcasters adopted a standard for the radio broadcast data system for stereo or mono FM, though AM may follow.

This quiet revolution applies to regular analog radio broadcasts. It does not require digital broadcasting, which is still several years in the future for the ordinary listener.

FM stations transmitting the Radio Data System (RDS) provide the opportunity for your receiver to display a variety of information-from station name to the names of songs. Dynamic receiver control is also possible, enabling auto retuning to the strongest transmitter from a chosen network, muting of cassette music for traffic announcements, and so on.

To see these messages, an RDS decoder and display are required with the tuner. Last year, the first domestic car radios were fitted with RDS, and several home-entertainment manufacturers released premium FM tuners with RDS.

A HISTORICAL CONTEXT

To find out how all this developed, we must go back to 1976. In that year, Swedish Telecom introduced a unique radio-paging system using the existing national network of public FM broadcast stations. A 57-kHz subcarrier was added to the regular stereo signal, 57 kHz being far enough away from the 38-kHz left-right subband not to bother the music.

A biphase-encoded digital signal contained paging-subscriber ID and phone number at the modest rate of 1187.5 bps. To ensure messages didn't get mislaid by FM-propagation fluctuations, a robust error-correcting and -detecting code was used. Since burst errors are more typical for radio propagation than random errors, the errors were corrected using single-burst, error-correcting, shortened-cyclic code [1].

Error-correcting codes were invented by Hamming and Shannon at Bell Labs shortly after WWII. These codes were applied extensively to rocket and satellite telemetry during the Cold War, where an error was of more than passing importance.

The particular code chosen by Swedish Telecom was a 26,16 code

Offset	Offset word $d_9, d_8, d_7 \ldots d_0$	Syndrome $S_9, S_8, S_7 \ldots S_0$
A	0011111100	1111011000
B	0110011000	1111010100
C	0101101000	1001011100
C'	1101010000	1111001100
D	0110110100	1001011000
E	*0000000000*	*0000000000*

$$H=\begin{pmatrix}
1000000000\\
0100000000\\
0010000000\\
0001000000\\
0000100000\\
0000010000\\
0000001000\\
0000000100\\
0000000010\\
0000000001\\
1011011100\\
0101101110\\
0010110111\\
1010000111\\
1110011111\\
1100010011\\
1101010101\\
1101110110\\
0110111011\\
1000000001\\
1111011100\\
0111101110\\
0011110111\\
1010100111\\
1110001111\\
1100011011
\end{pmatrix}$$

Table 1—*These are the syndromes and associated parity matrix used by the P/C for decoding the RDS information.*

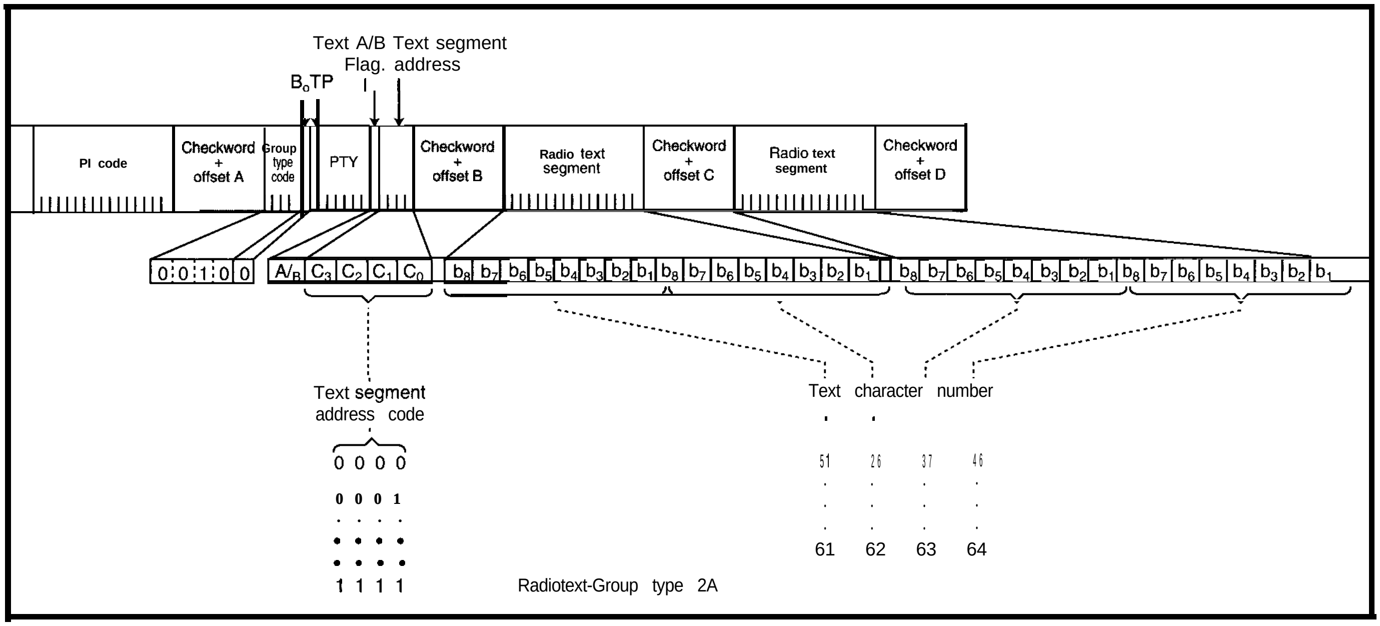


Figure l--Each group comprises 104 *bits split info four 26-bit blocks. The blocks are known* as *A,B,C,* or *D. The example* shown *here is Group 2A, used for radio text. Each block consists of 16 data bifs and 10 check bits.*

developed by Kasami. It is produced from the generator polynomial:

$$g(x) = x^{10} + x^8 + x^7 + x^5 + x^4 + x^3 + 1$$

This optimal error-correction’ code can detect an error-burst length of 5 using only 2 x 5 = 10 parity-check digits. A 26,16 code has 16 data digits and 10 parity-check digits in a 26-bit block.

To implement the code, the original 16-bit data vector is multiplied at the source by a 16 x 26 generator matrix G to give a 26-bit encoded block to broadcast. At the receiver, this block must be multiplied by a 26 x 10 parity-check matrix H to yield a 10-bit syndrome. If this syndrome matches the syndrome known to the receiver, synchronization may have been achieved (i.e., the block’s first 16 bits are correct).

The next three blocks are similarly handled. If four correct syndromes in a row are received, synchronization occurs. The data can finally be reduced to ASCII text or hex numbers.

In the early 1980s, the European Broadcasting Union selected Swedish MBS technology for development into RDS. Whereas MBS uses only the E (zero] syndrome for encoding, RDS uses deliberately introduced errors (offset words) to encode and identify four different blocks.

These blocks are always transmitted in groups, a sequence of 4 blocks or 104 bits. There are **16** possible types of

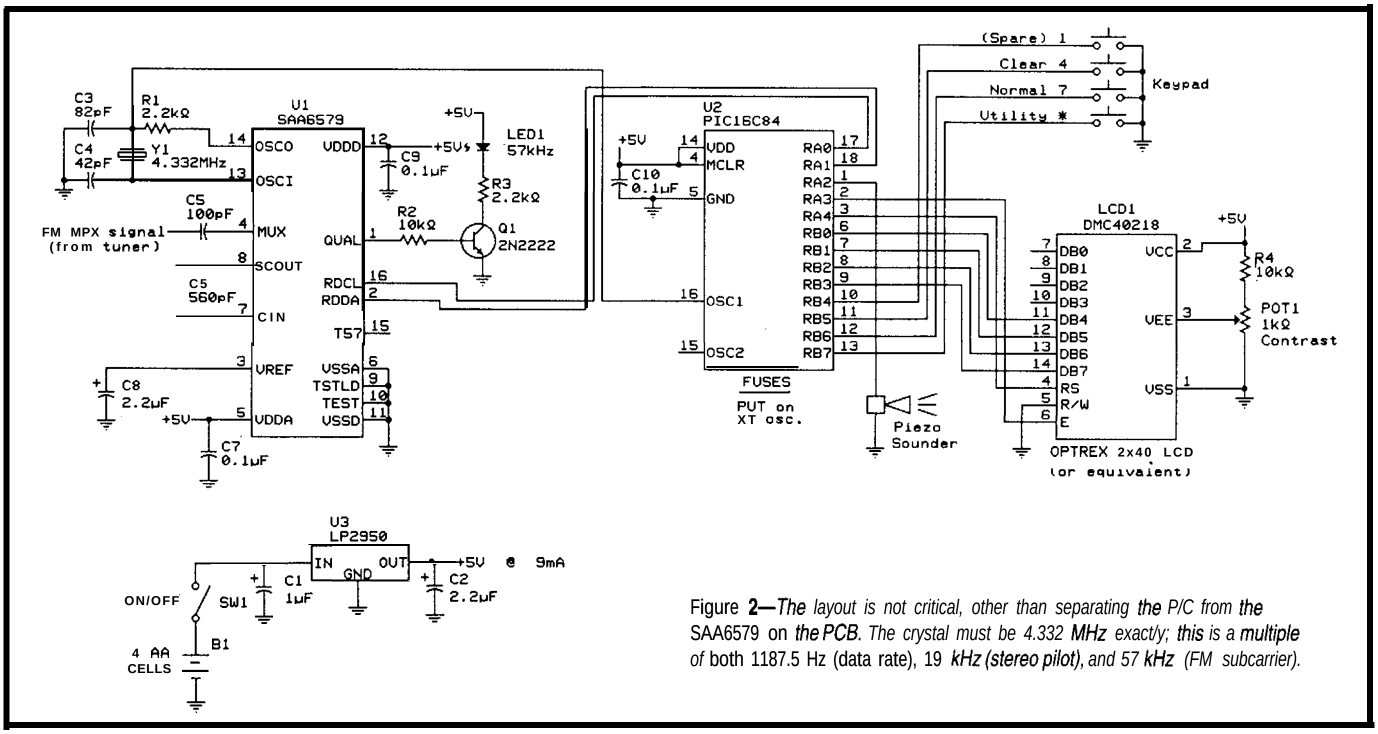


Figure 2—*The layout is not critical, other than separating the P/C from the* SAA6579 on *the PCB. The crystal must be 4.332 MHz exactly; this is a multiple of* both 1187.5 Hz (data rate), 19 *kHz (stereo pilot), and 57 kHz (FM subcarrier).*

Alabama			
	WZYP	104.3	Athens
	WRJM	93.7	Geneva
Arizona			
	KKFR	92.3	Phoenix
	KSLX	100.7	Scottsdale
California			
	KSIQ	96.1	Brawley
	KLON	88.1	LongBeach
	KPCC	89.3	Pasadena
	KTWV	94.7	LosAngeles
	KCRW	89.9	Santa Monica
	KHOP	104.1	Modesto
	KNPR	88.1	Ridgecrest
	KSFM	102.5	Sacramento
	KPBS	89.5	San Diego
	KPLM	106.1	Palm Springs
	KYXY	96.5	San Diego
	KDFC	102.1	San Francisco
	KEAR	106.9	San Francisco
	KALW	91.7	San Francisco
	KKSF	103.7	San Francisco
	KCBX	90.1	San Luis Obisco
Colorado			
	KMJI	100.3	Denver
	KCFR	90.1	Denver
Connecticut			
	WSHU	91.1	Fairfield
	WPKT	90.5	Hartford
	WTIC	96.5	Hartford
D.C.			
	WAMU	88.5	Washington
	WETA	90.9	Washington
	WDCU	90.1	Washington
	WGAY	99.5	Washington
Florida			
	WAPN	91.5	Daytona Beach
	WSFP	90.1	Fort Myers
	WAOA	107.1	Melbourne
	WLRN	91.3	Miami
	WTMI	93.1	Miami
	WMFE	90.7	Orlando
	WOCL	105.9	Orlando
	WUFT	89.1	Gainesville
	WOWW	107.3	Pensacola
	WFLZ	93.3	Tampa
Georgia			
	WSTR	94.1	Smyrna
	WABE	90.1	Atlanta
	WCLK	91.9	Atlanta
	WKLS	96.1	Atlanta
Illinois			
	WCIL	101.5	Carbondale
	WLRW	94.5	Champaign
	WPGU	107.1	Champaign
	WBEZ	91.5	Chicago
	WXRT	93.1	Chicago
	WLS	94.7	Chicago
	WNUA	95.5	Chicago
	WLLR	101.3	East Moline
	WAAG	94.9	Galesburg
	WSWT	106.9	Peoria
	WDBR	103.7	Springfield
	WGFA	94.1	Watseka
Indiana			
	WSHW	99.7	Frankfort
	WXKE	103.9	Fort Wayne
	WENS	97.1	Indianapolis
	WZPL	99.5	Indianapolis
	WITZ	104.7	Jasper
	WZWZ	92.7	Kokomo
	WWKI	100.5	Kokomo
	WLEZ	102.7	Terre Haute
Iowa			
	KRVR	106.5	Davenport
	KFMG	103.3	Des Moines
	KOEL	92.3	Oelwein
	KUOO	103.9	Spirit Lake
	KAYL	101.5	Storm Lake
Louisiana			
	WGGZ	98.1	Baton Rouge
	KFXY	96.7	Morgan City
	WLMG	101.9	New Orleans
	WMYZ	95.7	New Orleans
	KCIL	107.5	Houma
	KMJJ	100.1	Shreveport
Maryland			
	WHFS	99.1	Annapolis
	WEAA	88.9	Baltimore
	WXYV	102.7	Baltimore
	WETH	89.1	Hagerstown
Massachusetts			
	WBUR	90.9	Boston
	WGBH	89.7	Boston
	WBOQ	104.9	Gloucester
	WBCS	96.9	Newton
	WMJX	106.7	Boston
Michigan			
	WIOG	102.5	Bay City
	WLLZ	98.7	Detroit
	WKQI	95.5	Detroit
	WJLB	97.9	Detroit
	WQRS	105.1	Detr oit
	WDBM	88.9	EastLansing
	WKAR	90.5	East Lansing
	WLAV	96.9	Grand Rapids
	WRKR	107.7	Portage
Minnesota			
	KBEM	88.5	Minneapolis
	KNOW	91.1	SaintPaul
	KSJN	99.5	SaintPaul
Missouri			
	KYYS	102.1	KansasCity
Nebraska			
	KESY	104.5	Omaha
Nevada			
	KKLZ	96.3	Las Vegas
	KNPR	89.5	Las Vegas
	KNPR	88.7	Boulder City
	KNPR	91.7	Beatty
	KOMP	99.3	Henderson
	KNPR	88.7	Indian Springs
	KNPR	89.5	Laughlin
	KNPR	88.7	Moapa Valley
	KNPR	88.7	Pahrump
	KNEV	95.5	Reno
	KNPR	88.1	Scotty's Junction
	KNPR	105.1	Searchlight
	KLUC	98.5	Las Vegas
	KFMS	101.9	Las Vegas
	KEYV	93.1	Las Vegas
	KRRI	105.5	Las Vegas
	KOMP	92.3	Las Vegas
	KEDG	103.5	Las Vegas
	KFBI	107.5	Las Vegas
	KYRK	97.1	Las Vegas
	KLNR	91.7	Panaca
	KTPH	91.7	Tonopah
	KEYV	103.5	Laughlin
NewJersey			
	WFPG	96.9	Atlantic City
	WKDN	106.9	Camden
	WBGO	88.3	Newark
	WFME	94.7	Newark
	WNNJ	103.7	Newton
	WPAT	93.1	Paterson
	WADB	95.9	Point Pleasant
NewMexico			
	KKOB	93.3	Albuquerque
NewYork			
	WAMC	90.3	Albany
	WZRQ	102.3	Albany
	WHTZ	100.1	New York
	WNEW	102.7	New York
	WNYC	93.9	New York
	WBEE	92.5	Rochester
NorthCarolina			
	WUNC	91.5	Chapel Hill
	WCXL	104.1	Kill Devil Hills
NorthDakota			
	KNOX	94.7	Grand Forks
Ohio			
	WOUB	91.3	Athens
	WGUC	90.9	Cincinnati
	WVXU	91.7	Cincinnati
	WWNK	94.1	Cincinnati
	WCPN	90.3	Cleveland
	WGAR	99.5	Cleveland
	WENZ	107.9	Cleveland
	WKSU	89.7	Cleveland
	WLTF	106.5	Cleveland
	WLVQ	96.3	Columbus
	WDFM	98.1	Defiance
	WKRJ	91.5	New Philadelphia
	WKKO	99.9	Toledo
	WGTE	91.3	Toledo
	WKRW	89.3	Wooster
	WHIZ	102.5	Zainesville
	WOUZ	90.1	Zanesville
	WGLE	90.7	Lima
Oklahoma			
	KSYE	91.5	Frederick
	KIRQ	98.1	Lawton
Oregon			
	KYTE	102.7	Newport
	KOPB	91.5	Portland
	KKRZ	100.3	Portland
	KMCQ	104.5	The Dalles
Pennsylvania			
	WRTI	97.1	Allentown
	WITF	89.5	Harrisburg
	WRVV	97.3	Harrisburg
	WROZ	101.3	Lancaster
	WFLN	95.7	Philadelphia
	WHYY	90.9	Philadelphia
	WMMR	93.3	Philadelphia
	WRTI	90.1	Philadelphia
	WPLY	100.3	Philadelphia
	WXPN	88.5	Philadelphia
	WDUQ	90.5	Pittsburgh
	WRTI	97.7	Reading
RhodeIsland			
	WWBB	101.5	Providence
Tennessee			
	WYPL	89.3	Memphis
Texas			
	KEAN	105.1	Abilene
	KNLE	88.1	Austin
	KTTX	106.1	Brenham
	KTEX	100.3	Brownsville
	KKYS	104.7	Bryan
	KORA	98.3	Bryan
	KTEX	106.9	Bryan
	KAYD	97.5	Beaumont
	KQXY	94.1	Beaumont
	KYKR	95.1	Beaumont
	KCBI	90.9	Dallas
	KERA	90.1	Dallas
	KILT	100.3	Houston
	KUHF	88.7	Houston
	KOOI	106.5	Jacksonville
	KYKX	105.7	Longview
	KKMY	104.5	Orange
	KCRN	93.9	San Angelo
Utah			
	KSOS	106.9	Ogden
	KSOS	92.1	Salt Lake City
	KSOS	96.7	Salt Lake City
	KSOS	98.3	Utah County
Virginia			
	WLTY	95.7	Norfolk
	WNVZ	104.5	Norfolk
	WESR	103.3	Onley
	WKOC	93.7	Virginia Beach
	WCDX	92.7	Richmond
Washington			
	KFAE	89.1	Richland
	KUOW	94.9	Seattle
	KMPS	94.1	Seattle
	KRPM	106.1	Seattle
	KISW	99.9	Seattle
	KEZE	105.7	Spokane
Wisconsin			
	WERN	88.7	Madison
	WUWM	89.7	Milwaukee
	WMYX	99.1	Milwaukee

Table 2—*As of March 1995, there are RDS broadcast stations in 34 states and Washington DC. New stations are being added at the rate of 10% per month. The large number in Las Vegas can be attributed to the National Association of Broadcasters convention, held there every spring. There are about 6000 FM radio stations in the U.S.*

groups, each giving a different type of information (3 groups are reserved for future use). Table 1 defines the parity check matrix H and the syndromes that must be found in the datastream. Figure 1 shows a typical group.

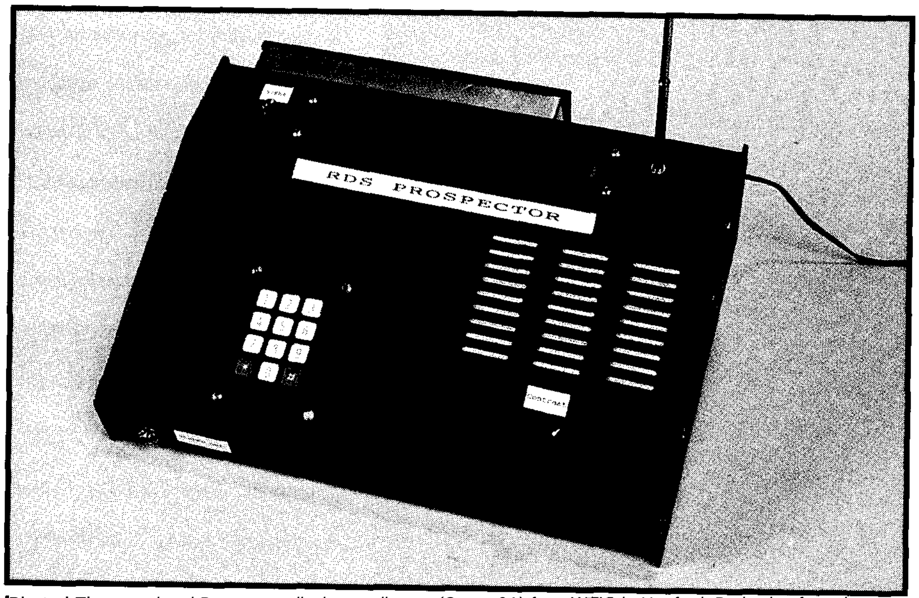
PROSPECTOR

Although the popularity of RDS is rapidly increasing in the U.S. (FM stations with RDS are doubling every nine months), there is a shortage of receivers on the market. To see the messages with an existing radio, a demodulator, microcontroller, and display are required. After some initial research, I designed and built the Prospector, my own combined unit. It seeks out all available groups.

Figure 2 presents the Prospector's schematic. To demodulate the biphase RDS data, which is present as a tiny (2% injection) 57-kHz signal (Figure 3), the sophisticated Philips SAA6579 is used. This chip has an 8-pole bandpass filter and Costas loop all on-chip. The raw multiplexed FM signal goes into the chip and RDS clock and data TTL signals come out (Figure 4).

For the microcontroller, the Microchip PIC16C84 was an easy choice. Its powerful Harvard architecture enables the critical 842-µs loop time for block processing to be met at the 4.332-MHz oscillator rates. The demodulator and microcontroller therefore can share the same crystal adding to the simplicity of design. As well, compared to the older EPROM technology of most micros, the EEPROM structure offers easy reprogramming of custom groups.

Lastly, let's consider the display. With 2 × 40 alphanumeric displays now below $10, there is little reason to scrimp. Curiously, many premium RDS receivers only display 8 uppercase characters at a time. I used a full 2 × 40 Optrex LCD for the 64-character radio text, leaving 16 characters for station name and clock time.



'Photo I-The *completed Prospector displays radio text (Group 2A) from WTIC in Hartford. Projecting from the rear is a Delco car radio.*

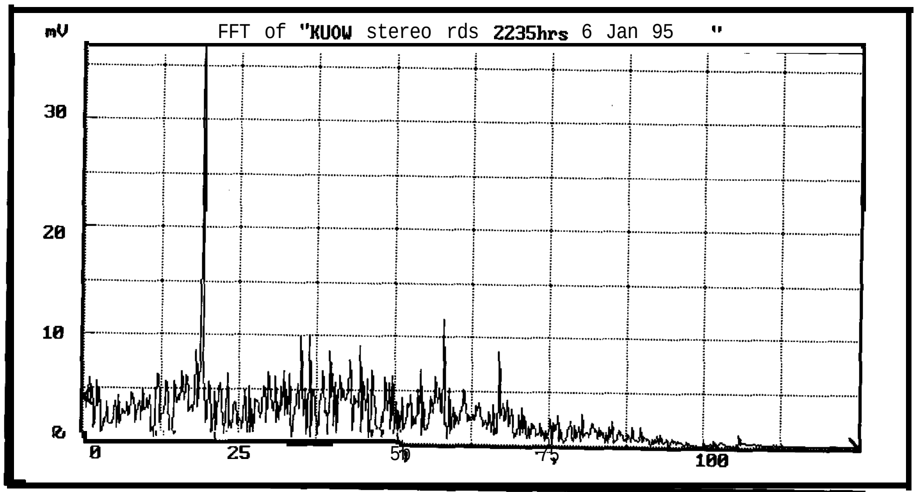


Figure 3—*This example is from a broadcast from KUOW (Seattle). Note the peaks at 19 kHz (stereo pilot), 57 kHz (RDS), and 67 kHz (SCA talking book service). It is also possible to send RDS via a mono signal.*

Group type						Applications
Decimal value	Binary code					
	A3	A2	A1	A0	B0	
0	0	0	0	0	X	Basic tuning and switching information
1	0	0	0	1	X	Program item number and slow labelling codes
2	0	0	1	0	X	Radiotext
3	0	0	1	1	0	LN-Location and Navigation
4	0	1	0	0	0	Clock-time and date
5	0	1	0	1	X	Transparent data channels (32 channels)
6	0	1	1	0	X	In-house applications
7	0	1	1	1	0	Radio paging
8	1	0	0	0	0	Reserved for TMC
9	1	0	0	1	0	Emergency warning systems
10	1	0	1	0	0	Program Type Name
11–13						Undefined
14	1	1	1	0	X	Enhanced other networks information
15	1	1	1	1	X	Fast basic tuning and switching information
X indicates that value may be "0" (version A) or "1" (version B).						

Table **3**—*RDS data is broken info groups, with fhe group type determining the kind of data if contains. There are 16 in all, of which 13 are currently in use. Many groups come in an A* or *B* variant.

MAGIC NUMBERS

The core program uses a subroutine Syndrome (shown in Listing 1) along with supporting routine Matrixhi. It takes about 600 µs to perform the multiplication of a 26-bit raw-data vector and a 26 x 10 parity matrix H. H is stored as 26 Retlw instructions in Matrixhi (highest 8 bits) and Matrixlo (lowest 2 bits).

Each bit of raw data is examined. If a 1 is found, the corresponding row is brought from H and exclusive ORed with its predecessor. Finally, the lo-bit result is compared to 1 of 5 known syndromes (see Table 1) to determine if a recognizable block has been found. If not, the next bit received is slipped into the 26-bit stack, which is held in four registers: Storeg 1-4. The whole process then repeats. Since another bit arrives every 842 µs, all matrix math must be finished in appreciably less time than this.

I experienced no trouble running a nominal 4-MHz 16C84 at 4.332 MHz. If necessary, a 4-MHz crystal can be added across pins 15 and 16 of the 16C84 as the code has been tested at both speeds. One especially nice feature of the 16C84 is the built-in pull-up resistors that can be enabled on Port B. This, together with the interrupt-on-change feature, makes for an elegantly simple keypad interface.

RADIO

A modern receiver is nice for rock-steady tuning, but is not essential. The FM multiplex signal is found after the FM-demodulator section, but before the stereo-decoder section on a modern FM tuner. On digitally tuned receivers, the search is made easier by the use of discrete ICs for each function.

If a schematic is not readily available, the best method is to wait for a quiet period on a classical station. Use a scope to look for the 19-kHz pilot tone as a sine wave of about 30-mV amplitude. Once the FM multiplex signal is fed to the SAA6579T, it is split into clock and data for input to the PIC (see Figure 5).

RDS IS OUT THERE

You don't have to search far for RDS. With the kind permission of John

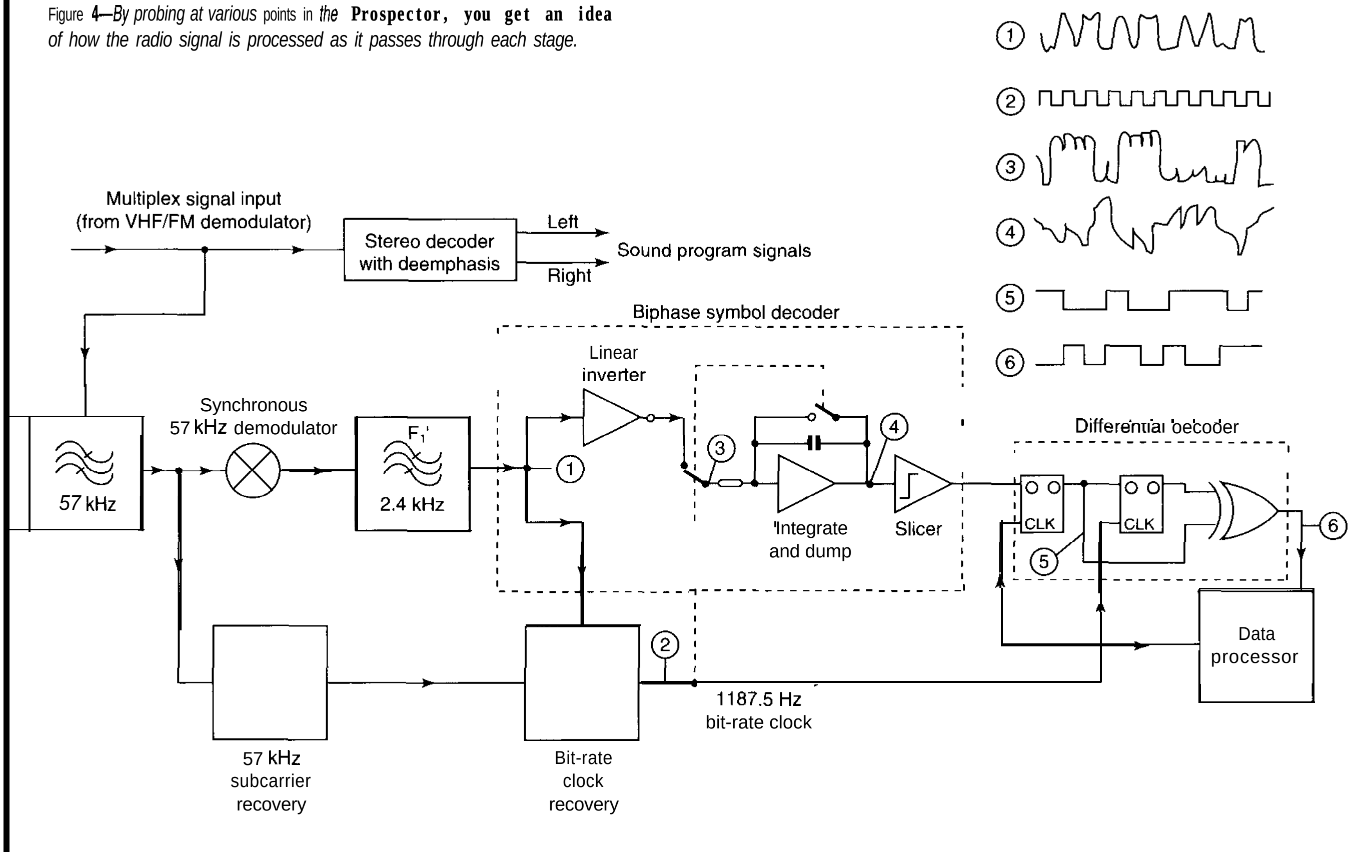


Figure 4—By probing at various points in the **Prospector, you get an idea** *of how the radio signal is processed as it passes through each stage.*

Gatski of *Radio world,* I have reproduced in Table 2 the location of all the RDS stations in the U.S. as of March 1995. As you can see, 34 states and Washington, DC are included. If your state isn't here, don't worry. RDS is soon coming to a station near you.

Here, in the Pacific Northwest, I can receive four Seattle-based RDS FM stations: KMPS, KUOW, KISW, and KRPM. Groups transmitted from these four include OA, OB, 1A, 2A, 4A, 6A, 1 OA, and 15A—enough to give all kinds of tuning, clock, and text information. KRPM is even using RDS to transmit differential GPS corrections to navigational users so they **can** gain much greater accuracy. Table 3 lists all groups currently in use.

Photo 1 shows a message received at ***INK's*** office from WTIC on the completed RDS Prospector. A substantial aluminum box houses the Prospector to provide room for a Delco FM1500 car radio and speaker. Automotive radios are usually far better designed than most consumer radios sinces the car is a harsh environment for radio reception.

In Europe, where I plan to take the RDS Prospector on a future trip, RDS has been in use for about 6 years and is widely used in newer car radios. About 75% of Western European FM stations are now RDS equipped.

Fortunately, U.S. and European RDS systems are compatible. Prospector should work on both sides of the Atlantic. However, there are minor differences. For example, program types (e.g., PTYN = 6) denotes drama in Europe and classic rock in the U.S.

CLOCKWISE

The software displays RDS clock time as 24-hour UTC (Universal Time Clock) with the offset to local time following. For example, "22:30 – 7" is equal to 10:30 P.M. Greenwich Mean Time less 7 hours or 3:30 P.M. Pacific daylight-saving time.

If a plus sign instead of a minus sign precedes the local offset, then you are no longer in North America. Most likely, you have encountered a misset station clock. RDS is still very new. In Europe, RDS clocks are often linked to an atomic time standard for split-second accuracy.

Because clock time on RDS is updated once per minute on the zero second, don't be surprised if you can't find the clock group (4A) with the utility program immediately.

USER INSTRUCTIONS

Assuming you've located the FM multiplex pickup point on your tuner, connect to the input of the RDS Prospector and switch both units on.

Since RDS is near audio frequency, there is little problem with circuit layout or interconnection to a radio. The sensitive RF section is not involved. As always, don't forget the bypass capacitors.

Cruise the dial until the 57-kHz LED glows brightly and steadily (it flickers dimly on nonRDS stations). Unless you've found an MBS station, text, station name, and clock time appear within seconds on the display. Use the RDS roll call in Table 2 to assist you, but don't forget that new stations are being constantly added.

If you have ever tried to interpret a noisy radio-teletype signal on the HF

Listing 1—*This code has to multiply the last 26 incoming data bits by a matrix to obtain the 10-bit syndrome before another bit arrives (i.e., in less than 842 µs).*

```
_MATRIXHI                        ; Contains RDS Matrix H 26 x 10
        addwf  PC,f              ; (upper 8 bits of each row
        nop                      ; with last row first etc.)
        retlw  0xC6
        retlw  0xE3
        retlw  0xA9
        retlw  0x3D
        retlw  0x7B
        retlw  0xF7
        retlw  0x80
        retlw  0x6E
        retlw  0xDD
        retlw  0xD5
        retlw  0xC4
        retlw  0xE7
        retlw  0xA1
        retlw  0x2D
        retlw  0x5B
        retlw  0xB7
        retlw  0x00
        retlw  0x00
        retlw  0x01
        retlw  0x02
        retlw  0x04
        retlw  0x08
        retlw  0x10
        retlw  0x20
        retlw  0x40
        retlw  0x80
. . .
_SYNDROME                        ; Multiplies the 26-bit block by
        movlw  0x1A              ; the 26x10 RDS Matrix H to produce
        movwf  COUNTREG          ; a 10-bit Syndrome
        clrf   HIREG             ;Hireg/Loreg eventually contain
        clrf   LOREG             ; the Syndrome
        movf   STOREG4,w
        movwf  STOREG4X
        movf   STOREG3,w
        movwf  STOREG3X
        movf   STOREG2,w
        movwf  STOREG2X
        movf   STOREG1,w
        movwf  STOREG1X
_LOOPS
        rlf    STOREG4X,f        ; Shuffle stack by 1 bi
        rlf    STOREG3X,f
        rlf    STOREG2X,f
        rlf    STOREG1X,f
        btfss  STATUS,CARRY; Was it a '1'?
        got0   _LOOPX            ; For a '0'
        movf   COUNTREG,w
        call   _MATRIXHI         ; Get approp. row from Matrix
        xorwf  HIREG,f           ; and exclusive OR it
        movf   COUNTREG,w
        call   _MATRIXLO         ; Do same for last 2 bits of Matrix row
        xorwf  LOREG,f
        decfsz COUNTREG,f
        got0   _LOOPS            ; Repeat 26 times in less than 842 µs
        return
_LOOPX                           ; This is trap for 0, no exclusive OR
        movlw  0x03              ,just padding to avoid reading the
        call   _DELAY            ;  same bit twice
        decfsz COUNTREG,f
        got0   _LOOPS
        return
```

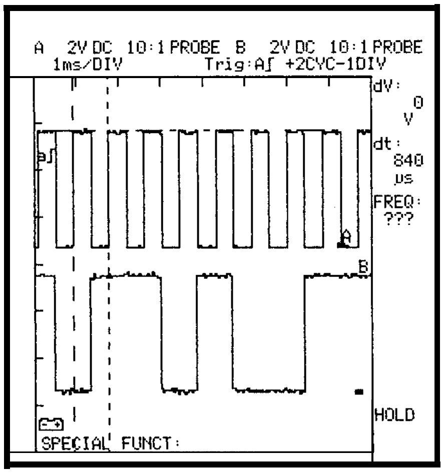


Figure 5—*This* signal, from an actual broadcast from *KMPS (Seattle),* becomes the input to the *PIC.* The upper trace is the clock and the lower is data.

band, RDS's complete freedom from displayed errors will amaze you. Cyclic codes really do work!

Press the Utility key and the group types found (eight samples) will be displayed along with traffic information flags, PTY, PI code, and the music or speech bit. Clock time and station identity (PS) continue to be displayed.

To sample another eight group types (stations typically repeat a sequence of groups frequently), press Utility again. Pressing Normal returns you to the text message. Any time you want the display erased, press Clear. Otherwise, the display latches the last message.

A piezo sounder provides a 1 -kHz beep for every key press of an inexpensive membrane keypad. One spare key is wired into RB4 for future options.

FINALE

The intriguing possibilities of the new Radio Data System are now yours to explore. Radio is still a magical medium, even 60 years after Major Edwin Armstrong's classic invention of FM broadcasting. ❑

Christopher Morris has an M.Sc. in railway engineering. He is currently maintenance engineer for the advanced light rapid-transit vehicles at Skytrain in Vancouver, British Columbia. He may be reached at (604) 520-3641.

REFERENCE

[1] Shu Lin, ***An Introduction to Error-Correcting Codes*** (Englewood Cliffs, NJ: Prentice-Hall, 1970) 185–201.

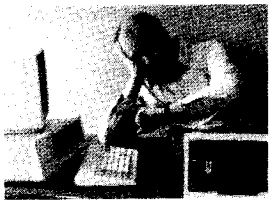
SOURCE

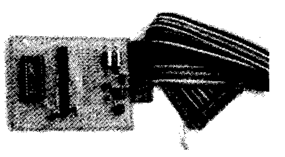
RBDS Standard
National Association of Broadcasters
1771 N. Street NW
Washington, DC 20036-2891
(202) 4295373
Fax: (202) 7753515

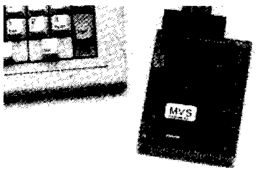
Audio-Radio Data Handbook
Philips Semiconductors Literature
1000 Business Center Dr.
Mount Prospect, IL 60056
(708) 296-5461
Fax on demand: (800) 282-2000

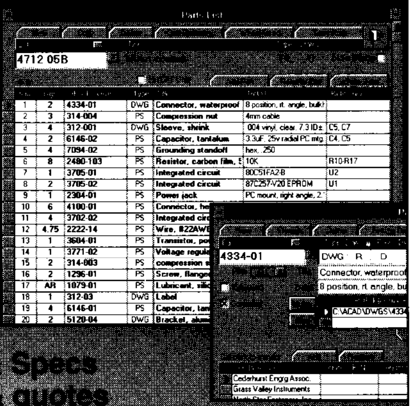
I R S

404 Very Useful
405 Moderately Useful
406 Not Useful









4712 05B

1	2	4334-01	DWG	Connector, waterproof	8 position, rt. angle, bulkh	
2	3	314-004	PS	Compression nut	4mm cable	
3	4	312-001	DWG	Sleeve, shrink	.004 vinyl, clear, 7.3 ID±	C5, C7
4	2	6146-02	PS	Capacitor, tantalum	3.3uF, 25v radial PC mtg.	C4, C5
5	4	7094-02	PS	Grounding standoff	hex, .250	
6	8	2480-103	PS	Resistor, carbon film, 5	10K	R10-R17
7	1	3705-01	PS	Integrated circuit	80C51FA2-B	U2
8	2	3705-02	PS	Integrated circuit	87C257-V20 EPROM	U1
9	1	2304-01	PS	Power jack	PC mount, right angle, 2.1	
10	6	4100-01	PS	Connector, he		
11	4	3702-02	PS	Integrated circ		
12	4.75	2222-14	PS	Wire, #22AWG		
13	1	3604-01	PS	Transistor, po		
14	1	3771-02	PS	Voltage regula		
15	2	314-003	PS	compression s		
16	2	1296-01	PS	Screw, flange		
17	AR	1079-01	PS	Lubricant, silic		
18	1	312-03	DWG	Label		
19	4	6146-01	PS	Capacitor, tan		
20	2	5120-04	DWG	Bracket, alumi		

4334-01
DWG R D
Connector, waterproof
8 position, rt angle, bu
C:\ACAD\DWGS\4334

Cederhurst Engrg Assoc.
Grass Valley Instruments

FEATURE ARTICLE

Kenneth Ciszewski

Siemens ESCC2 UART

Malting a High-Speed Audio Link Sing

Not at the speed of light, but almost the speed of sound, the ESCC2 sings. This UART offers the harmony of high-speed asynchronous transmission through hardware interface, initialization, and software algorithms.

Serial data communication is taken for granted in today's highly computerized world. Getting its start by connecting modems and terminals to mainframes and minicomputers, the serial interface is used to connect personal computers to devices ranging from touchscreens to industrial controls. In general, these interfaces are somewhat limited. Although some serial interfaces claim to run at 115 kbps, 9.6-19.2 kbps are the common upper bounds of data rates.

Still, there are times when even that isn't fast enough. For instance, the transmission of digitized audio/voice information in real time is even more demanding.

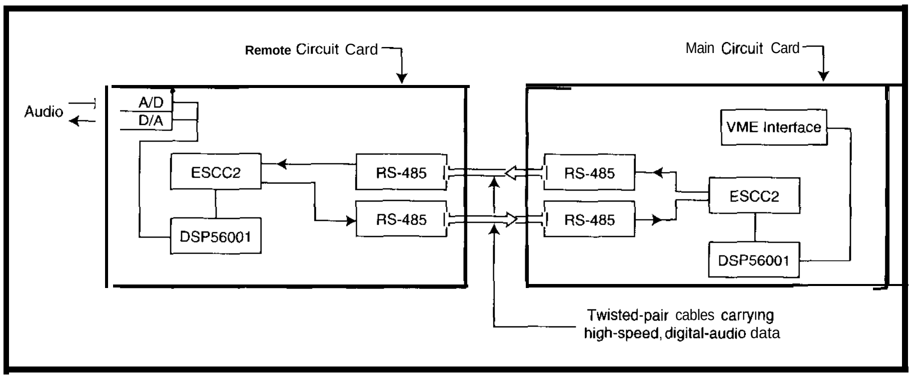


Figure 1—The *digital-audio* system uses *high-speed data transmission to meet real-time communications requirements.*

Various ways have been used to transmit digital voice data over serial data links. The telephone companies have been doing it with T1 circuits for years. The coming of ISDN (Integrated Services Digital Network) offers some additional choices. There are also various methods of connecting digital telephones to local PBX equipment. It is even possible to use Ethernet and FDDI to transmit digital voice from place to place.

Unfortunately, to manage the data transceiver, these methods require considerable resources both in hardware (the number of integrated circuits) and software (computer time spent managing the hardware). As a result, they take up considerable circuit-card space and are costly. Further, some of the data formats are specialized and limited, and many of the interfacing integrated circuits are expensive or hard to obtain.

After a lot of searching, a high-performance UART-style device turned out to be the effective and compact choice.

This article describes the use of the Siemens ESCC2 UART to transmit serial data over distances of up to 1000' at burst rates of 768 kbps

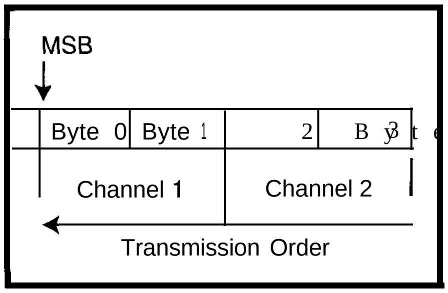
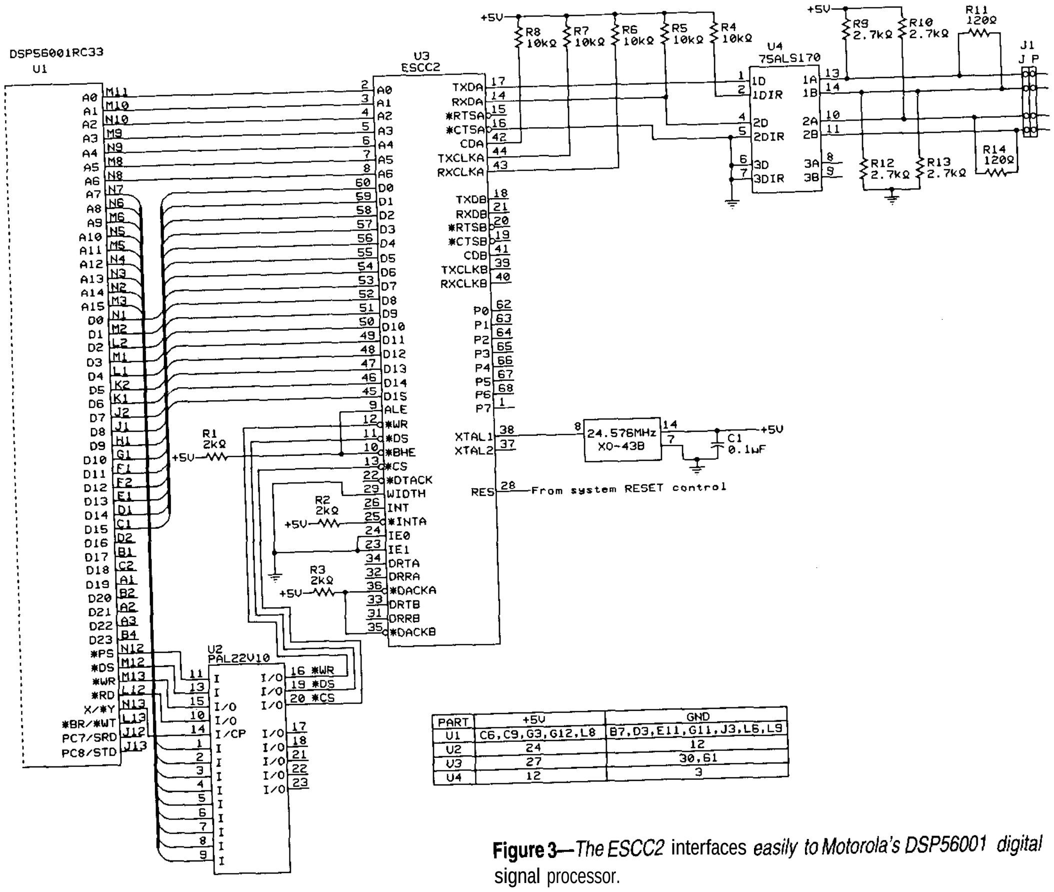


Figure 2—*Two channels of digital-audio data are transmitted in big endian format.*



PART	+5V	GND
U1	C6,C9,G3,G12,L8	B7,D3,E11,G11,J3,L6,L9
U2	24	12
U3	27	30,61
U4	12	3

Figure 3—*The ESCC2* interfaces *easily to Motorola's DSP56001 digital* signal processor.

.**isting 1**—*One* programmable *logic* device *does all the decoding* required *by the ESCC2.*

```
  Logic Symbols
  "/"  = NOT
  "+"  = OR
  "*"  = AND

:Inputs to decoder PAL are outputs of DSP56001
\15-A7 ;address signal lines
RD     ;read signal line
WR     ;write signal line
PS     ;program memory space signal line
IS     ;data space signal line
XNY     :X-Y memory space signal line

;Outputs from decoder PAL to inputs on ESSC2
R_W    ;read/write signal line
CS_U   ; chip select signal line
DS_U   ;data strobe signal line

;ESCC2 UART
;transmit FIFO is at Y:$C000-$C03F
;receive FIFO at $C000-$C03F

/CS_U=A15* A14 * /A13 * /A12 * /A11* /A10 * /A9 * /A7 * /XNY * /DS

;ESCC2 UART data strobe
/DS_U=/WR + /RD

;ESCC2 UART read/write select line
/R_W=A15 * A14 * /A13 * /A12 * /A11 * /A10 * /A9 * /A8 * /XNY * /D'S
```

(effective rate is 256 kbps). Details of the hardware interface, ESCC2 initialization, and software algorithms show how very high-speed asynchronous transmission may be achieved with the ESCC2.

The software algorithms are described in generic terms rather than in the specific assembly language of the DSP56001. Although pseudocode is impossible to immediately implement, I chose it because it eases porting the code to other processors.

DESIGN REQUIREMENTS

I chose the Siemens ESCC2 as the means of sending digital-audio data from a remotely located circuit card to a main circuit card where digital-audio processing takes place. As Figure 1 shows, the processed digital audio is then sent back to the remote circuit card. Two digital audio channels of 2 bytes each (4 bytes total) must be sent each direction 8000 times per second (i.e., once per digital-audio frame) (see Figure 2).

Since 4 bytes is equal to 32 bits, the required minimum effective data rate (for each direction) is:

32 bits per frame x 8000 frames per second = 256 kbps.

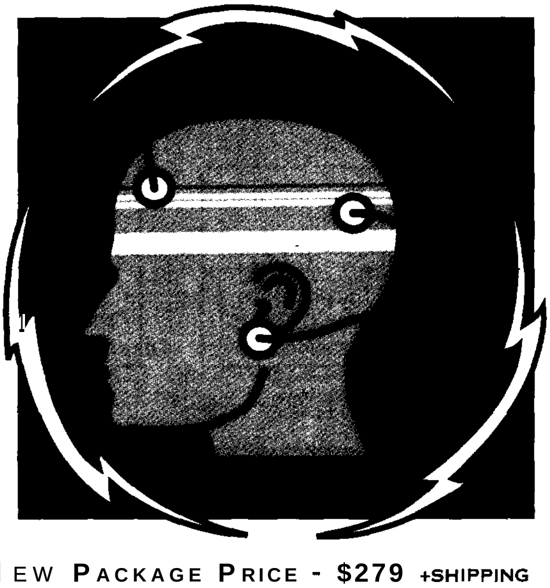
This is well within the capability of the ESCC2's maximum data rate of 2 Mbps.

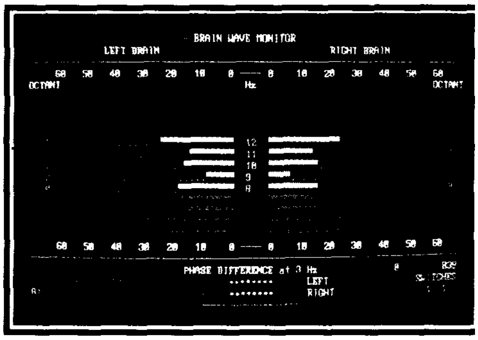
To achieve data transmission at distances of up to 1000′, I used the EIA RS-422 transmission standard. RS-422 provides differential point-to-point data communication at distances up to 4000′, depending on the data rate.

Name	Location	Description
RFIFOA	Base + $00	Ch A read FIFO starts (2-32 bytes)
RFIFOB	Base + $40	Ch B read FIFO starts (2-32 bytes)
XFIFOA	Base + $00	Ch A trans FIFO starts (2-32 bytes)
XFIFOB	Base + $40	Ch B trans FIFO starts (2-32 bytes)
STARA	Base + $20	Ch A Stat Reg (R/O)
STARB	Base + $60	Ch B Stat Reg (R/O)
CMDRA	Base + $20	Ch A Cmd Reg (W/O)
CMDRB	Base + $60	Ch B Cmd Reg (W/O)
MODEA	Base + $22	Ch A Mode Reg (R/W)
MODEB	Base + $62	Ch B Mode Reg (R/W)
TIMRA	Base + $23	Ch A Tmr Reg (R/W)
TIMRB	Base + 563	Ch A Tmr Reg (R/W)
TCRA	Base + $26	Ch A Termination Char Reg (R/W)
TCRB	Base + 566	Ch B Termination Char Reg (R/W)
DAFOA	Base + 527	Ch A Data Format Reg (R/W)
DAFOB	Base + 567	Ch B Data Format Reg (R/W)
RFCA	Base + 528	Ch A Rec FIFO Ctrl Reg (R/W)
RFCB	Base + $68	Ch B Rec FIFO Ctrl Reg (R/W)
RBCLA	Base + 52A	Ch A Rec Byte Count Low Reg (R/O)
RBCHA	Base + $2B	Ch A Rec Byte Count High Reg (R/O)
RBCLB	Base + $6A	Ch B Rec Byte Count Low Reg (R/O)
RBCHB	Base + $6B	Ch B Rec Byte Count High Reg (FUO)
XBCLA	Base + 52A	Ch A Trans Byte Count Low Reg (W/O)
XBCHA	Base + $2B	Ch A Trans Byte Count High Reg (W/O)
XBCLB	Base + 56A	Ch B Trans Byte Count Low Reg (W/O)
XBCHB	Base + $6B	Ch B Trans Byte Count High Reg (W/O)
CCROA	Base + $2C	Ch A Config Reg 0 (R/W)
CCR1A	Base + $2D	Ch A Config Reg 1 (R/W)
CCR2A	Base + $2E	Ch A Config Reg 2 (R/W)
CCROB	Base + $6C	Ch B Config Reg 0 (R/W)
CCR1B	Base + $6D	Ch B Config Reg 1 (R/W)
CCR2B	Base + $6E	Ch B Config Reg 2 (R/W)
VSTRA	Base + 534	Ch A Ver Stat Reg (R/O)
VSTRB	Base + 574	Ch B Ver Stat Reg (R/O)
BRGA	Base + 534	Ch A Baud Rate Generator (W/O)
BRGB	Base + 574	Ch B Baud Rate Generator (W/O)
GIS	Base + $38	Global Int Stat (R/O)
IVA	Base + 578	Int Vector Address (W/O)
IPC	Base + $39	Int Port Config
ISR0A	Base + 53A	Ch A Int Stat Reg 0 (R/O)
ISR0B	Base + 57A	Ch B Int Stat Reg 0 (R/O)
ISR1A	Base + $3B	Ch A Int Stat Reg 1 (R/O)
ISR1B	Base + $7B	Ch B Int Stat Reg 1 (R/O)
IMR0A	Base + 53A	Ch A Int Mask Reg 0 (W/O)
IMR0B	Base + 57A	Ch B Int Mask Reg 0 (W/O)
IMR1A	Base + 53B	Ch A Int Mask Reg 1 (W/O)
IMR1B	Base + 57B	Ch B Int Mask Reg 1 (W/O)
PVR	Base + 53C	Port Value Reg (R/W)
PIS	Base + 53D	Port Int Stat Reg (R/O)
PIM	Base + 53D	Port Int Mask Reg (W/O)
PCRA	Base + 53E	Port Config Reg (R/W)

Note: Differentiate registers appearing to overlap by R/W signals.

Table 1—*Memory-mapped registers are used to configure and control the ESCC2.*





Listing 2—The ESCC2 is initialized for asynchronous mode, 8 data bits, 1 start bit, 1 stop bit, no parity, and 768 kbps. Register names used are defined in Listing 1. Channel B is initialized the same way as Channel A.

```
Start "Initialization"

  Move $00-> CCROA
  Move $00-> MODEA
  Move $00-> CCR1A
  Move $00-> CCRPA
  Move $41-> CMDRA   Reset channel A transmit and receive fifos
  Move $ff-> IMROA   Disable interrupts
  Move $ff-> IMR1A   Disable interrupts

[Initialize for Asynchronous Mode, 8 data bits, 1 start bit,
1 stop bit, no parity. 768 kbps]
  Move $03-> CCROA   Powerdown in asynchronous mode
  Move $1f-> CCR1A   Output = push-pull
                     Asynchronous mode
                     bit rate
                     clock mode 7
  Move $38-> CCRPA   oscillator-> baud rate generator->
                     receive clock and transmit clock
  Move $00-> BRGA    baud-rate divisor = divide by 2
  Move $fe-> IMROA   enable receive pool full interrupt
                     (mask out all other interrupts)
  Move $4c-> IMR1A   enable interrupts: all sent, transmit pool
                     ready, break detected
  Move $04-> RFCA    set read FIFO threshold to 4 bytes
  Move $03-> IPC     set interrupt outputs active high
  Move $08-> MODEA   turn on channel receiver
  Move $00-> DAFOA   8 data bits, 1 stop bit, no parity,
                     break = normal
```

(continued)

Because it is necessary to send 32 bits in both directions in less than the 125 µs allowed for a digital-audio frame, the full-duplex arrangement was necessary.

The ESCC2 is connected to a Motorola DSP56001 digital signal processor at both ends of the RS-422 transmission line. The DSP56001 provides digital-audio processing on the main circuit card and an interface to the analog-digital and digital-analog converters at the remote circuit card end.

HARDWARE DESIGN

The ESCC2, depicted schematically in Figure 3, appears to a controlling microprocessor as a pair of data FIFOs (transmit and receive) and a series of configuration and control registers. A chip-select (*CS) signal is generated by decoding a selected base address from the high-order microprocessor address lines. Address lines A0–A6 are connected directly to the ESCC2 to address individual data and register locations.

Listing **2—*continued***

```
  Move $83—> CCROA    powerup, asynchronous mode, NRZ encoding
  Move $f0—> PCRA     set 8-bit port: upper 4 bits = input
                      lower 4 bits = output
  Move $ff—> PIM      Mask 8-bit port interrupt
  Move $41—> CMDRA    Reset channel A transmit and receive FIFOs

[Test status reg. to be sure reset is complete before continuing.]
  Test (bit CEC of STARA)
    If(=1), wait until (= 0) to continue,
      else continue.
End "Initialization"
```

This implementation uses the ESCC2 Motorola M68000 bus interface mode, which is selected by setting the ALE pin high. This mode requires a single read/write signal (.WR) and a data strobe (● DS) signal.

The DSP56001 external bus provides separate read (*RD) and write (* WR) signals, but no data strobe (*DS) signal. It also provides signals that tell which memory space is being accessed externally: P (program), X (data), or Y (data). As a result, it is quite a bit different from the M68000 bus. AMD's PALCE22V10H-10PC/5 was chosen to create the required control signals for the ESCC2. Listing 1 includes the logic equations for programming the PAL.

The DSP56001 uses a 33-MHz clock, which means that a zero-wait-state access to the ESCC2 takes place in 60.6 ns. However, the timing specifications for the ESCC2 indicate that it cannot be accessed that quickly. The DSP56001's external access to memory can be extended by programming 1-15 wait states. Two extra wait states provide correct access timing.

Although the ESCC2 provides a hardware interrupt to signal the arrival of received data, I didn't use it in this design since digital-audio processing is based on a synchronous 8-kHz sampling rate used to interrupt the DSP56001. The operations of the ESCC2 are effectively synchronized by the DSP56001.

Listing 3-Four *bytes of data are written to the **ESCC2's FIFO** buffers for* transmission during *each 125-µs* frame.

```
"Turn-Break-Signal-Off"

Start "Turn-Break-Signal-Off"
  Move $00—> DAFOA Turn off break signal on transmit
                   data output line
End "Turn-Break-Signal-Off"

"Write-to-DART-FIFO"
Start  "Write-to-DART-FIFO"

[Writes 4 bytes to Ch A Xmit FIFO and commands ESCC2 to send data.1
  [reset Ch. A transmit FIFO]
  Move  $01—> CMDRA1
  [Test status reg to be sure reset is complete before continuing.1
  Test (bit CEC of STARA)
    If (= 1), wait until (= 0) to continue,
      else continue.
  [Wait for Ch. A ISR1 interrupt (which is pending).]

GTEST Test (bit ICAO of GIS1)
  If (= 0), wait until (= 1) to continue,
    else continue.

[Check that Ch. A transmit FIFO is ready for new data.]
Test (bit XPR of ISR1)
  If (= 0), go to GTEST
  If (= 1), continue.
  Move four bytes of data—> Ch. A transmit fifo.
  [Transmit Ch. A data.]
  Move $08—> CMDRA
End  "Write-to-DART-FIFO"
```

CHOICE OF TRANSMISSION MODE

I chose to use the asynchronous transmission mode of the ESCC2 for this design. This mode transmits data bytes using a start bit, eight data bits, and a stop bit in the same manner as RS-232 transmissions from personal computers. The difference here is that the ESCC2 transmits using this method at much higher data rates than most UARTs can support.

The ESCC2 internally divides a system clock signal to provide transmission and reception baud-rate clocks. In clock mode 7, the ESCC2 automatically divides the external clock by 16. An additional "divide-by-two" can be programmed using the Channel Configuration Register 2 and the Baud Rate Generator Register.

In this design, an external 24.576-MHz clock signal is provided. Dividing this clock by 32 (16 x 2) gives a 768-kbps transmission rate. This enables 40 bits (32 bits + 2 start/stop bits per byte x 4 bytes) to be transmitted in 40 x 1.3 us or 53 µs, which is just under one-half an 8-kHz frame time of 125 µs.

ESCC2 INITIALIZATION

Table 1 lists the ESCC2 registers and their functions. Listing 2 gives the initialization sequence for channel A of the ESCC2 (channel B is configured similarly). The ESCC2 is initialized in clock mode 7 for asynchronous operation at 768 kbps.

SOFTWARE DESIGN

In addition to Listing 2's initialization sequences, I developed four other software routines for the DSP56001. Listing 3 shows Turn Break Signal Off and Write to UART FIFO, Listing 4 presents Read from UART FIFO, and Listing 5 includes Transmit Break. In the actual implementation, after the one-time initialization, the DSP56001 waits for an 8-kHz frame-rate interrupt. After it receives this, it consecutively calls Turn Break Signal Off, Write to UART FIFO, Read From UART FIFO, and Transmit Break.

TESTRESULTS

Initially, I wrote the software and tested the system without the use of the Transmit Break and Turn Break Signal Off routines. Although data was transmitted and received at the 768 kbps rate, there were problems.

Occasionally, the byte order appeared to scramble at the receiver, causing annoying noise in the listener's headsets (it sounded like surf crashing on the beach at about 110 db!). Also, if one of the remote circuit cards (as many as four remote circuit cards can be connected to one main card) was turned off and then on, the system would get lost, producing more noise.

It was clear that there needed to be some way to "frame" each four bytes of data. I hadn't made any provision to synchronize the frame rates of the various circuit cards because the design uses analog-to-digital converters that set the interrupt rate of each remote circuit card.

I solved the problem with the transmit break and the break interrupt register signals. After four bytes of data are loaded into the ESCC2's transmit FIFO, the software routine checks to see if all four bytes have been sent out. If they're gone, the Transmit Break command is sent to the ESCC2, which forces its transmit data (Txd) output from its normal high (or space) state to a low (or mark) state. Txd remains low until it is time to send another four bytes in the next frame. Txd is then brought high before actual serial data transmission begins.

The receiving ESCC2 detects the break signal on the line. After reading the four bytes received out of its FIFO, it resets the receive FIFO and waits for the next four bytes to arrive.

An actual system using the techniques described here was in-

Listing 4—*Data* reception is *simplified by using the ESCC2's "receive pool full" flag.*

```
"Read-from-UART-FIFO"
[Reads four bytes from Ch. A (packed as 2 words).]

Start "Read-from-UART-FIFO"
[Check for receive pool full flag = 1)

  Test (bit ICAO of GIS1)

    If (= 1), continue,
    else if (= 0) go to BRKDET.
  [Test to see if receive FIFO is full up to threshold.]

  Test (bit RPF of ISR0)
    If (= 0), exit [go to BRKDET],
    else if (= 1), continue.
  Move four bytes of data from Ch. A receive FIFO.

  [Verify receive pool has been completely read.]
    Move $80-> CMDRA.

  [Look for pending interrupt.]

BRKDET Test (bit ICA1 of GIS)
  If (= 0), exit.
    Else if (= 1)
      Test (bit BRK of ISR1A)
        If (= 1) Exit
          else if (= 0) continue

    Test (bit CEC of STARA)
      If (= 1), wait until (= 0).

  [Reset Receiver and Receive FIFO.1

  Move $40-> CMDRA
  Exit

End  "Read-from-UART-FIFO"
```

Listing 5—*Data transmission is framed by using the ESCC2's break signal feature.*

```
"Transmit Break"
[Transmits break signal on serial data transmit line after all
data has been sent out of transmit FIFO.]

Start "Transmit Break"
  [Wait for pending Ch. A Interrupt.]

BRK Test (bit ICA1 of GIS)
  If (= 0), test again.
    Else continue.

[Check to see if Transmit FIFO is empty1
  Test (bit ALLS of ISR1)
    If (= 0), go back to BRK
      Else continue.

  [Transmit break command.]
  Move $40-> DAFOA

End "Transmit Break"
```

stalled in a customer facility. The distance between the main circuit card and the farthest remote circuit card was about 200'. The system has performed satisfactorily since December 1993.

CONCLUSION

As this application demonstrates, the ESCC2 makes very high-speed serial data transfer practical and inexpensive over reasonable distances.

While this implementation was for digital audio, the concept can be extended to other systems such as point-of-sale terminals and industrial controls. Packets of up to 16 bytes could be transferred with minimal microprocessor intervention. ❑

Ken Ciszewski has been designing electronic equipment for communications systems for more than 20 years. He was project manager for a computer-controlled voice recording and paging system for the Lambert-St. Louis Airport. More recently, he has been designing digital audio systems for networked flight trainers. He may be reached at ciszewsk%adadv1.decnet@mdcgwy.mdc.com.

SOURCES

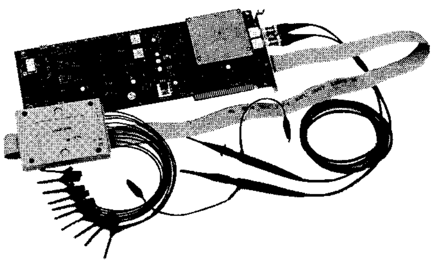
ESCC2 **(SAB82532N10-V22)**
Siemens Components, Inc.
10950 North Tantau Ave.
Cupertino, CA 95014
(408) 777-4500
Fax: (408) 777-4958

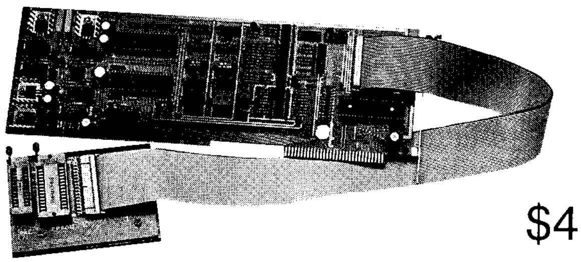
DSP56001
Motorola, Inc.
(602) 952-4103
Fax: (602) 952-4067

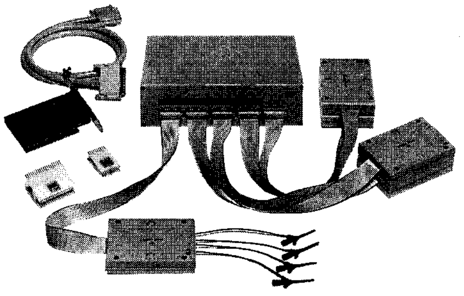
PALCE22V10H-10PC/5
Advanced Micro Devices, Inc.
901 Thompson Pl.
P.O. Box 3453
Sunnyvale, CA
(408) 732-2400

I R S

407 Very Useful
408 Moderately Useful
409 Not Useful







FEATURE ARTICLE

Robert Priestley

Low-Cost PC-based Universal 68HC705 Development System

Faced with a mammoth project, Robert realized he'd never get done unless traditional burn methods were replaced with something less tedious. His solution: a 68HC705 development system.

Ohis project started with a request from friends to make a "computer thingie" to monitor the performance of their car and assist with navigation in an Australian outback car rally. The car computer was intended to track fuel, distance, time, speed, and RPMs. Obviously, my friends' primary goal was to finish the course first without getting lost or running out of fuel 1000 km from the nearest 7-Eleven.

The objectives seemed clear. However, when I initially tried to undertake the project, I found commercially available development tools expensive and hard to get. Without the tools, the standard development process proved to be too tedious. I realized that the only way I'd complete my car computer project was to first create a 68705 development system.

In this article, I present the 68705 development system that I designed. I'll review basic features of the board before introducing you to its programming characteristics.

BASIC FEATURES

To develop code with the 68705 development system, all you need is a DOS text editor and a cross-assembler so the program can be written and developed on a PC. Fortunately, Motorola offers a freeware cross-assembler with full documentation.

Once you have successfully assembled your program, you need to test it. I developed a simulator package for the PC which supports all of the features of the microcontroller. This saved a great amount of time since software simulation is more versatile and quicker than burning EPROMs and trying to figure out what went wrong in the 68705!

Finally, you need programming hardware so the developed program can be downloaded to the EPROM in the 68705.

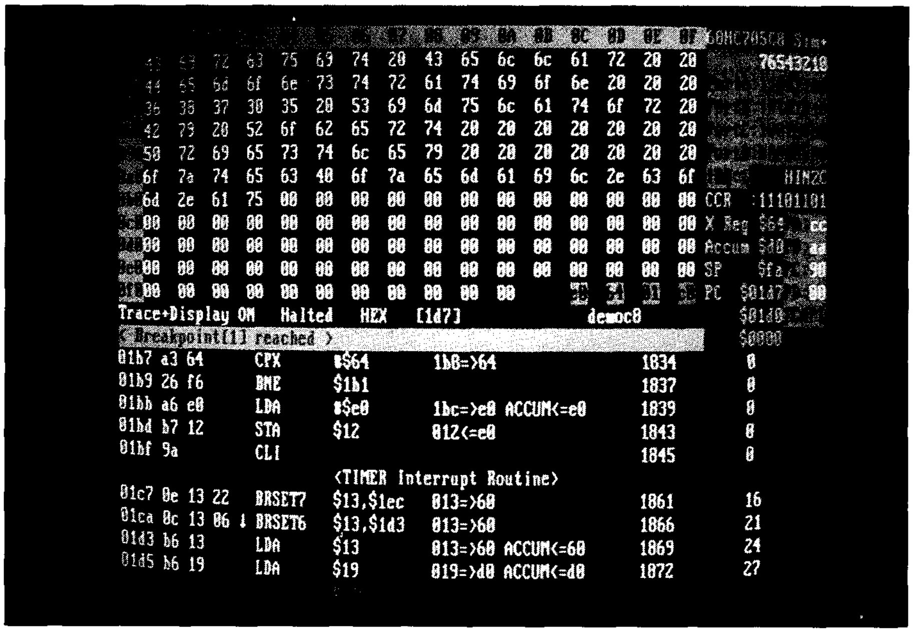
This development system has two parts-hardware and software. The hardware is contained on a professional-quality, double-sided, plated-through, screen-printed PCB which connects to an PC-compatible computer. The PCB is therefore easy to construct. And, because it connects to the parallel and serial ports of a PC, you don't need special interface cards to support it.

The complete software package includes the freeware Motorola 68705 cross-assembler, a file to operate the programmer board (68PROG05. EXE), a simulator (68SIM05. EXE), a disassembler (68DIS05. EXE), monitor programs, and demonstration programs. In other words, the development package has everything you need to develop an application for the 68705 microcontroller series.

IN-CIRCUIT EMULATION

The programmer board has an emulation socket for the 68HC705C8 which can be plugged into a target system under development. This socket can be converted by use of small adaptor boards so that other controllers can be emulated by the C8 controller.

In-circuit emulation (ICE) is achieved by two methods. The first gives limited real-time emulation of a program running in a C8 controller. With this method, you add an interrupt-driven monitor routine (68C8 MON.ASM)totheendofthesource code. When a command is sent from the PC or a breakpoint is reached, the C8 controller services the interrupt-



```
                                     07  08  09  0A  0B  0C  0D  0E  0F 68HC705C8 Sim+
 43  53  72  63  75  69  74  20  43  65  6c  6c  61  72  20  20        76543210
 44  65  6d  6f  6e  73  74  72  61  74  69  6f  6e  20  20  20
 36  38  37  30  35  20  53  69  6d  75  6c  61  74  6f  72  20
 42  79  20  52  6f  62  65  72  74  20  20  20  20  20  20  20
 58  72  69  65  73  74  6c  65  79  20  20  20  20  20  20  20
 6f  7a  74  65  63  40  6f  7a  65  6d  61  69  6c  2e  63  6f            HINZC
 6d  2e  61  75  00  00  00  00  00  00  00  00  00  00  00  00 CCR    :11101101
 00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00 X Reg  $64     cc
 00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00 Accum  $d0     aa
 00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00 SP     $fa     90
 00  00  00  00  00  00  00  00  00  00  00                     PC   $01d7     00
Trace+Display ON   Halted   HEX   [1d7]               democ8         $01d0
< Breakpoint[1] reached >                                            $0000
01b7 a3 64       CPX     #$64       1b8=>64                  1834     0
01b9 26 f6       BNE     $1b1                                1837     0
01bb a6 e0       LDA     #$e0       1bc=>e0 ACCUM<=e0        1839     0
01bd b7 12       STA     $12        012<=e0                  1843     0
01bf 9a          CLI                                         1845     0
                         <TIMER Interrupt Routine>
01c7 0e 13 22    BRSET7  $13,$1ec   013=>60                  1861     16
01ca 0c 13 06 ↓  BRSET6  $13,$1d3   013=>60                  1866     21
01d3 b6 13       LDA     $13        013=>60 ACCUM<=60        1869     24
01d5 b6 19       LDA     $19        019=>d0 ACCUM<=d0        1872     27
```

Photo 1—*The simulator can be used to examine* memory, set breakpoints, *and watch program flow.*

monitor routine. The RAM, registers, and I/O status of the controller are transmitted to the PC for evaluation OK modification.

The second ICE method enables any supported controller to be simulated through software on the PC while the hardware is emulated by a C8 controller. The controller executes a special monitor program (68C8 I C S . S 19) which communicates with the simulator package via the serial port. There are some minor hardware limitations to this method since there is no ADC for the R3, only one interrupt line, and no SPI or SCI hardware emulation for the C8 chip.

HCMOS AND HMOS TECHNOLOGY

The “HC” designation in the 68HC705 series signifies that it is a high-speed, low-power CMOS version. The HC chips contrast with the HMOS devices, which are relatively power hungry and slow.

The HCMOS series has a wide variety of variants (over 70). Table 1 shows the main features of the most popular microcontrollers (both HCMOS and HMOS) supported by the development system. From this table, you can select a microcontroller which suits your design requirements. There is also support for others not listed.

One point to note about Motorola microcontrollers is the “P” and “S” designations placed at the end of the part number (e.g., 68HC705C8P). “P” indicates that the controller is a one-time programmable device (OT-PROM). Typically, these devices are only used after a program has been finalized, such as in a production run. Because these devices are packaged without quartz windows and are manufactured by the truckload, they are considerably less expensive than the “S” EPROM devices, which are used for program development and can be erased by a UV light source.

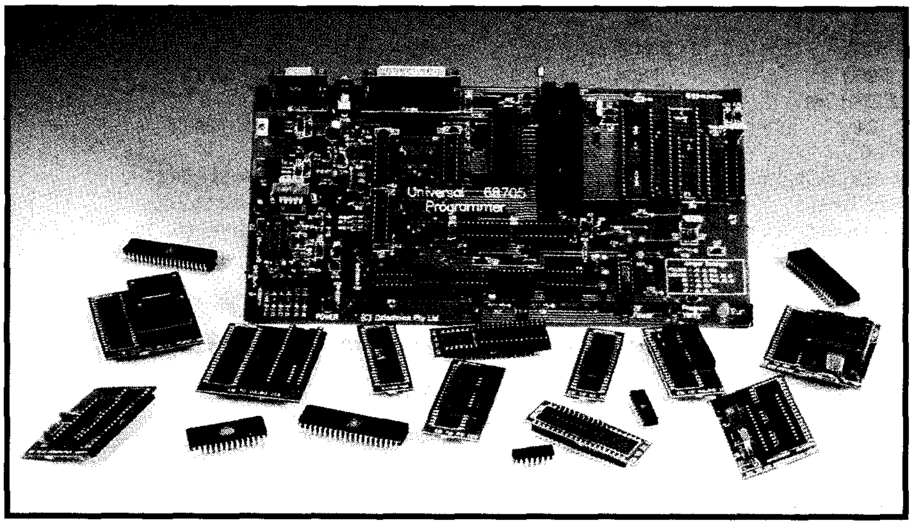


Photo 2—*Programmer* adapter boards add **support** for the *P9, El, and* ***B5 microcontrollers.*** *In-circuit simulation* adapter boards add ICS support for *K1, J2,* P3, *U3, R3,* and *P9* controllers.

PROGRAMMING METHODS

Motorola uses three methods to program their microcontrollers. To program the HMOS devices (and the 68HC705J2), you need two I/O lines of the controller to reset and clock the external address counter. The external memory device (containing the program to be burned in) is accessed by an 8-bit I/O port (Port A).

The 68HC705C8 controller uses Port B to read the external memory device while Ports A and C address the external memory device. This versatile device also has a serial port which can be used to up- and download programs.

The 68HC705K1 contains a bootstrap program.The PC programs the K1 controller by running a program across the parallel printer port to access the internal registers and bootstrap code of the K1. The programmer board and software supports both reading and writing to the K1 EPROM.

The printer port is used in a bidirectional mode with this controller to transfer data to and from the K1 controller. If you intend to use this controller, you need to make sure that your printer port operates in a bidirectional or enhanced mode. You can test this with the program 68PROG05. EXE.

Since the microcontrollers have different programming voltages (V,,), it is necessary to select the correct voltage by using jumper settings on the programmer board (these are clearly marked). Besides having to set these jumpers physically, the programmer board’s functions are completely

	MC68HC705C8, C4	MC68HC705J2	MC68HC705K1	MC68HC705B5
Technology	HCMOS	HCMOS	HCMOS	HCMOS
Number of Pins (DIL)	40	20	16	56
On-chip RAM (bytes)	176-304	112	32	176
On-chip EPROM (bytes)	7600-7744	2064	504	6208
Personality EPROM (bytes)	0	0	64 (bits)	0
Bidirectional Lines	24	14	10	24
Unidirectional Lines	8	0	0	8
I/O Features	SCI, SPI, 16-bit timer	15-bit timer withinterrupt	15-bit timer with interrupt	8-ch ADC SCI, 16-bit timer with interrupt, input capture, pulse-length DAC (PLM)
External Interrupt Input	1	1	4	1
Power Saving Stop, Wait, and Data-Retention Modes	yes	yes	yes	yes
Computer Operating Properly	yes	yes	yes	yes
Software Programmable Interrupt Sensitivity	yes	yes	yes	yes
EPROM secure mode	yes	no	no	yes
Emulation	MC68HC705C8, C4	MC68HC705J1		MC68HC705B6, B4

Table 1a—*The* number of Motorola microcontrollers *and* their mixture of features give *the designer the opportunity to select exactly fhe right processor for each* application.

automated by the PC via the parallel port. There are no messy switches and timing sequences to follow.

THE SOFTWARE PACKAGE

The software controlling the programmer board (68PROG05. EXE] performs many functions:

- display program dump
- provide information on number of PROM bytes and the percentage of PROM used
- offers built-in, troubleshooting capabilities
- program, verify, and secure PROM
- up- and download program
- execute program in RAM / PROM
- monitor C8 controller.

Some of these options are not available on all of the controllers.

Whenrunning 68PROG05.EXE, you specify a <file>.S19 program file using standard Motorola data format for assembled programs and the controller type (e.g., P3, U3, R3, K1, J2, C4, C8). The program automatically selects the correct menu options to display for the controller specified. The program also checks the .S19 file to ensure that it relates to the specified microcontroller. If anomalies are found, they are reported, saving you from loading inappropriate code.

As well, you can manually control the programmer board for testing purposes. This control is achieved by manipulating the bits in the data and control ports of the printer port while monitoring the status port.

The simulator package (68SIM05.EXE) lets you dry run your programs on a PC before you commit them to silicon (see Photo 1). You can step line by line through a program, undo previously executed instructions, or run to a breakpoint. All of the various

	MC68HC805B6	MC68705P3, P5	MC68705U3, U5	MC68705R3, R5
Technology	HCMOS	HMOS	HMOS	HMOS
Number of Pins (DIL)	56	28	40	40
On-chip RAM (bytes)	176	112	112	112
On-chip EPROM (bytes)	5952	1804	3776	3776
Personality EPROM (bytes)	56 EEPROM	0	0	0
Bidirectional Lines	24	20	24	24
Unidirectional Lines	8	0	8	8
I/O Features	8-ch ADC SCI 16-bit timer with interrupt, input capture, pulse-length DAC (PLM)	E-bit counter prescaler	8-bit counter prescaler	4-ch ADC, 8-bit counter with prescaler
External Interrupt Input	1	1	2	2
Power Saving Stop, Wait, and Data-Retention Modes	yes	no	no	no
Computer Operating Properly	yes	no	no	no
Software Programmable Interrupt Sensitivity	yes	no	no	no
EPROM secure mode	yes	no/yes	no/yes	no/yes
Emulation	MC68HC705B6, B4	MC68705P2, P4, P6	MC68705U2, U3	MC68705R2, R3

Table 1 b-The HCMOS *controllers offer the* additional benefit of very low *power* requirements compared *to the HMOS parts.*

registers, memory locations, and I/O can be viewed and modified. The source code and simulator trace screen can be viewed on screen. Display values can be formatted in hexadecimal, decimal, or as ASCII characters.

The simulator also traps stack overflows and illegal instructions. As well, writes to RAM locations used by the stack and programs, or writes to an illegal address (such as a read only address) can be trapped.

The 68HC705C8 monitor program (68C8MON.ASM) is an interrupt-driven routine tacked onto the end of your development program. Programs under development can be controlled from a PC by inserting breakpoints using software interrupt instructions (SWI).

When the controller finds an SWI, it jumps to the monitor routine where it dumps the status of the microcontroller back to the PC via the serial port. While in monitor mode, you can modify the various registers, I/O, and RAM. Once finished, the monitor routine can be exited using RTI, and the program continues from where it left off. The monitor routine for the PC is a function of the programmer file (68PROG05.EXE).

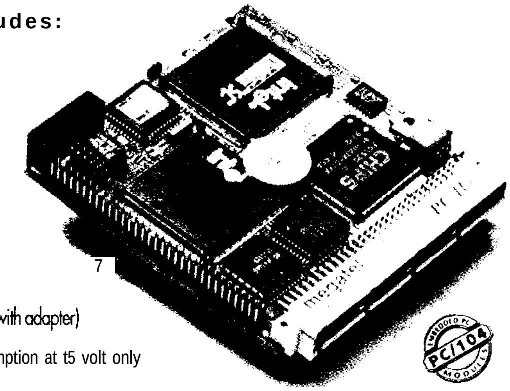
The 68C8ICS.S19 program enables the C8 controller to interface via the serial port to the simulator to perform hardware emulation.

THE PROGRAMMER BOARD

The programmer board can be expanded to burn any 8-bit microcontroller device such as the 68HC705E1, 68HC705B5, and 68HC705P9 by making a suitable extender board.

As you can see from Photo 2, the programmer board accommodates two 40-pin ZIF sockets for the C8, C4, U3, U5, R3, and R5; one 28-pin ZIF socket for the P3 and P5; one 20-pin ZIF socket for the J2; and one 16-pin ZIF socket for the K1 microcontroller. Note that the HMOS 68705 P, U, and R sockets overlap to save board space (you can only program one device at a time any way!).

The circuit of the programmer is shown in Figures 1 and 2. The circuit consists of a power supply, a DC voltage converter, RAM, address counters, tristate buffers, control



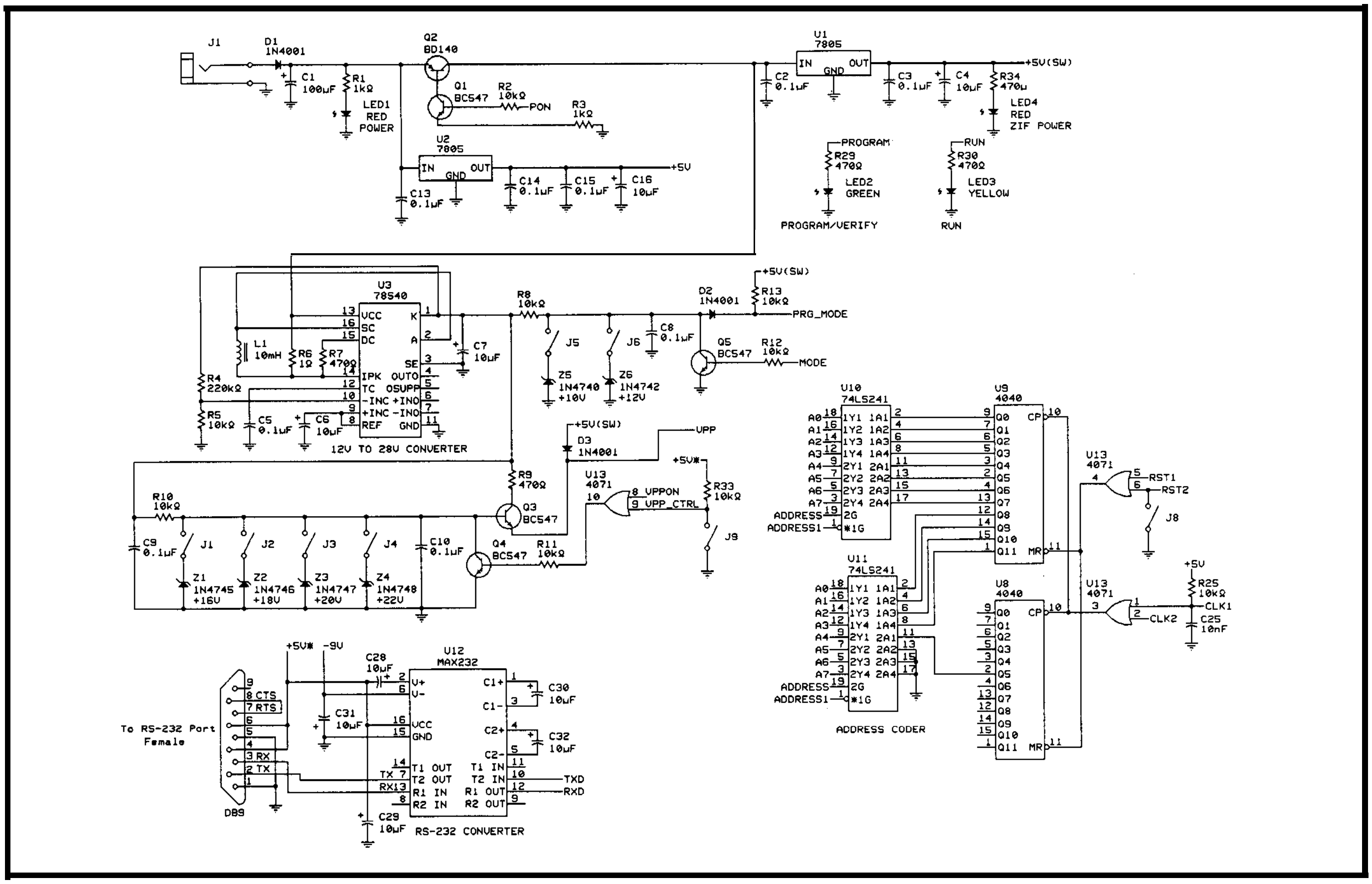
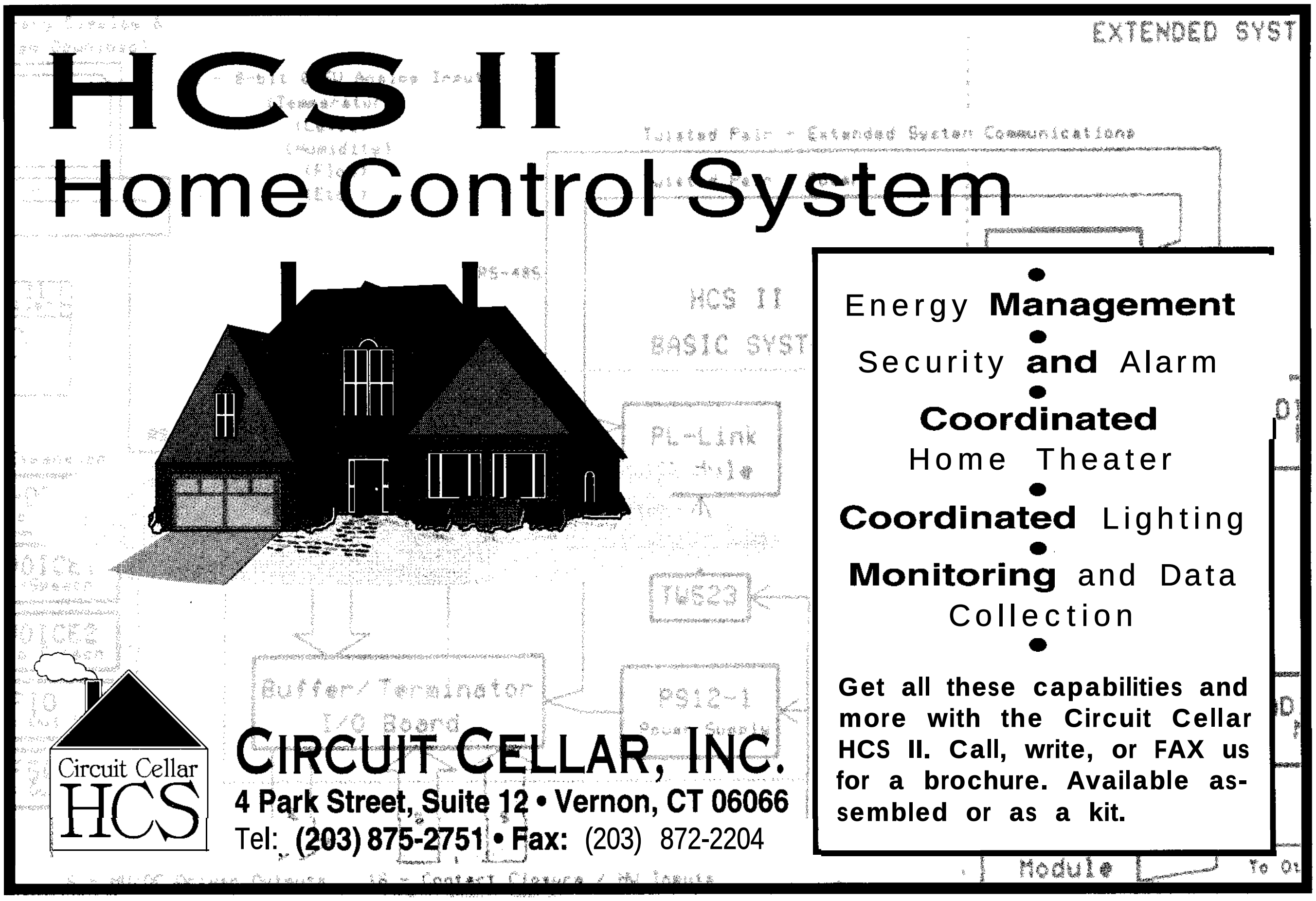


Figure 1a—*Jumpers are used to set the proper programming voltage depending on which processor is being used.*



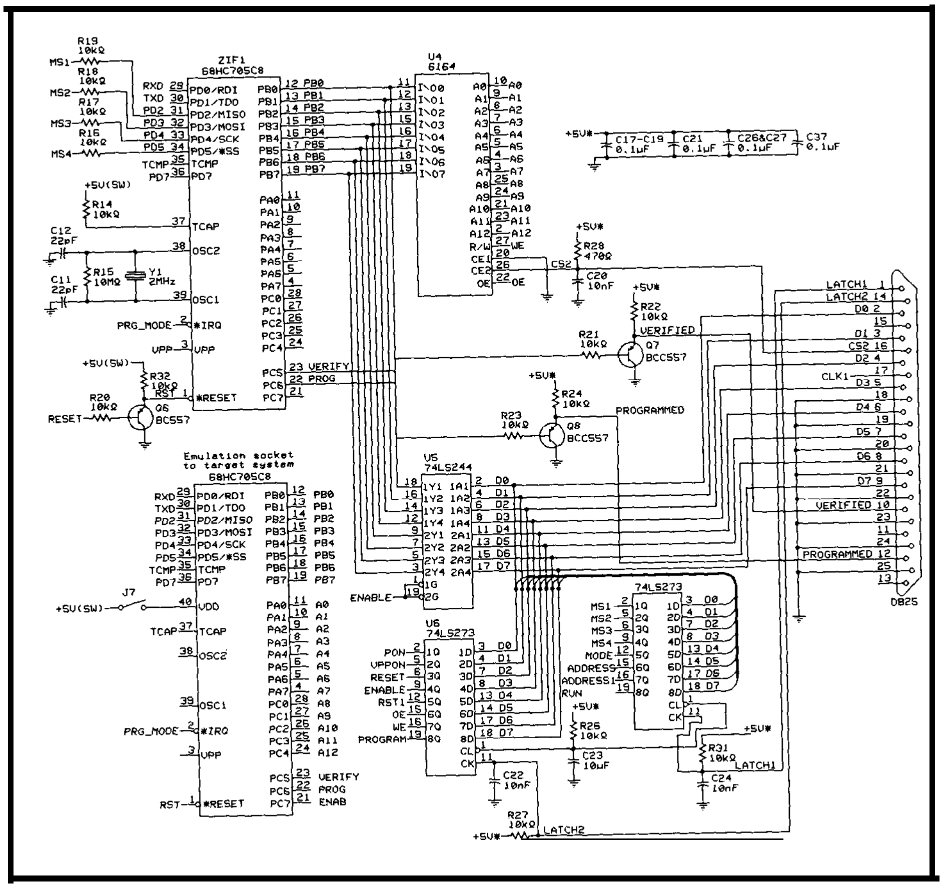


Figure 1b—*The* programmer board includes a *parallel interface to the PC printer port,* data *latches, main C4 and C8* programming socket, and ICE connection.

circuitry, serial interface, and two transistor buffers which interface the controller's signals to the printer port.

The programmer board connects to a parallel printer port of a PC-compatible computer via a standard DB25 cable. The usual eight data lines transfer data to and from the programmer. Six additional lines are used as control and status lines.

The C8 controller can also be connected to a PC's serial port. This interface enables the more advanced features of the C8 controller to be implemented. Data can be read from the controller or a program can be downloaded to the C8 controller's RAM, RS-232 conversion between the PC and C8 controller is implemented by U12, an ICL232 level shifter.

Control of the programmer board is achieved by two octal latches (U6 and U7) that connect to the printer port. Pin 1 of the port is pulsed (low-high-low) to latch data into U7. Pin 14 is pulsed to latch the data into U6.

Output Q1 of U6 (pin 2) switches the transistor switch formed by TR1 and TR2. Power to the ZIF sockets is indicated by LED L4. Note that you should not insert or remove a controller when this LED is on.

Output Q2 of U6 (pin 5) controls the programming voltage V_{PP} to the input of the microcontroller via the circuit formed around TR3, TR4, and U13c. When this output is a high, TR4 is turned on via U13c, an OR gate, which switches TR3 and V_{PP} off. Diode D3 then supplies +5 V to the V_{PP} terminal of the ZIF sockets. When TR3 is on [and TR4 off), it supplies the programming voltage to the ZIF sockets. The V_{PP} voltage depends on which zener (Z1-Z4) has been selected via jumpers (Jl-J4). This arrangement enables different V_{PP} voltages to be selected for different microcontrollers.

Output Q3 of U6 (pin 6) resets the microcontroller. A low at this output turns on TR6, pulls the RESET line of the microcontroller low, and holds the controller in a reset state.

Output Q4 of U6 (pin 9) enables buffer This buffer isolates the printer port from the 6264 RAM (U4) during a read cycle.

Output Q5 of U6 (pin 12) resets the address counters formed by US and U9 via U13b.

Output Q6 of U6 (pin 15) enables the RAM (U4) output during a read cycle. Output Q7 (pin 16) selects a read or write operation for u4.

Output Q8 of U6 (pin 19) is used to drive L2, the program and verify indicator LED.

Outputs Ql-Q4 of U7 (pins 3,4,7, and 8) select what bootstrap mode the C8 controller executes. Some available modes include: program and verify PROM, verify PROM contents, secure PROM contents and verify, secure PROM contents and dump, load program into RAM and execute, and dump PROM contents.

Output Q5 of U7 (pin 12) places the C8 controller into (or out of) bootstrap mode. If this output is high, TR5 is on, which removes the bootstrap voltage the controller recognizes. This step is taken after a C8 controller has been programmed. The controller can be left in the ZIF socket, the programmer board connected to the target system via the emulation socket, and the program run and monitored via the serial port. Note jumpers J5 and J6 select the bootstrap voltages that the controllers require.

Output Q6 and Q7 of U7 (pins 15 and 16) are complementary. They enable the output of the address decoders U8 and U9, depending on which microcontroller is being used.

Because the C8 controller has an 8-KB address range, it is necessary to connect U9 (4040) and U8 (4020) in parallel to cover the full addressing range of the controller. The outputs of the address decoders are buffered via U10 and U11 tristate buffers. With the C8 controller, these buffers are initially enabled when downloading a program from the PC into the RAM of the programmer board. When the C8 is programmed, these buffers are disabled since the C8 controller uses its own I/O ports to address the RAM of the programmer.

U3 and its associated components are used to produce a 28-VDC output from the 12-VDC input. The 28-V supply is regulated by R10 and the appropriate zener diode (Z1–Z4) to supply the correct V_{PP} voltage to the ZIF sockets.

The transistor buffers formed by TR7 and TR8 are connected to the status port bits 6 and 5 of the printer port. These two inputs indicate the status of the controller (programmed and verified].

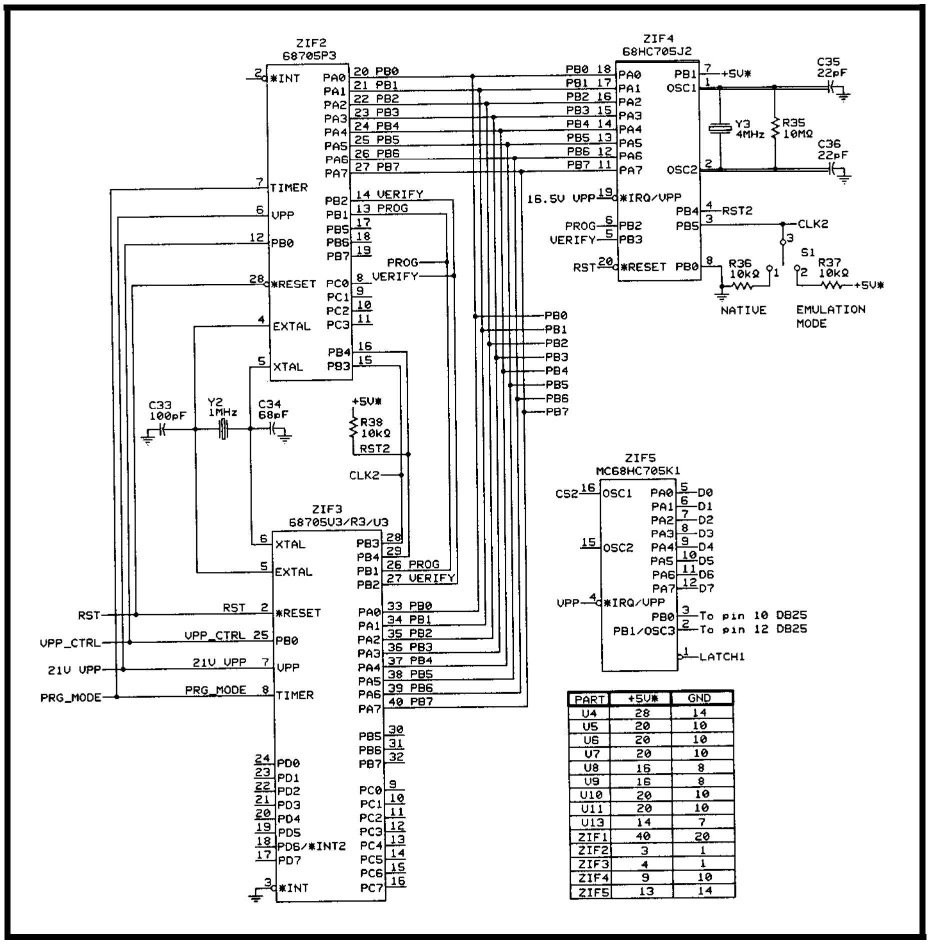
PROGRAMMING SEQUENCE

The first step in programming the C8 controller is to check that its internal PROM is erased. Erasure is achieved by filling the programmer's RAM with Os, then running the controller's bootstrap program. If the internal PROM is blank, the controller reports that it has been successfully programmed and verified (i.e., all bits in the internal PROM are 0).

The next step is to download the program from the PC into the 6264 RAM. To do this, the outputs of the address counters U8 and U9 have to be reset to 0, U10 and U11 outputs have to be enabled, U5 output has to be enabled, write has to be selected on the 6264 RAM, and power to the ZIF sockets turned off. To do this, latch the appropriate data bytes into U6 and U7.

Once the programmer board has been initialized, the first data byte (address $000) is sent out of the printer port. This data byte is written to the 6264 RAM by pulsing the CS2 line (pin 16 of the printer port). The next address in RAM is selected by clocking the address counters via control line CLK2 (pin 17 of the printer port). This process is repeated until all memory locations have been cycled through.

The next instruction the PC sends to the programmer board is to turn the programmer on; hold the controller in reset; apply the programming voltage to the V_{PP} pin; disable the output of U5, U10, U11; and enable the output of the U4. Remember that the C8 uses its own I/O ports for addressing the external memory device whereas the HMOS devices (and the 68HC705J2) require an external address counter.



PART	+5V*	GND
U4	28	14
U5	20	10
U6	20	10
U7	20	10
U8	16	8
U9	16	8
U10	20	10
U11	20	10
U13	14	7
ZIF1	40	20
ZIF2	3	1
ZIF3	4	1
ZIF4	9	10
ZIF5	13	14

Figure 2—*Many different processors are handled by using custom adapter boards that account for fhe unique features of each.*

The next instruction the PC sends to the programmer board is to release the reset line on the controller by making this line a high, hence turning off TR6.

Once the C8 controller comes out of reset, the bootstrap program takes over. An address location is loaded onto the address lines (ports AO-A7 and CO-C4) and the RAM contents read by Port B. The next address is selected and the byte is read. This process is repeated until all locations have cycled through.

Once all locations in the PROM have been burned, PC6 is pulled low indicating the end of the programming. The process is repeated again to compare the internal contents of the PROM to the external memory device (6264). If all checks out, PC5 pulls low to indicate a successful burn.

HMOS PROGRAMMING

For the HMOS (and the J2) devices, the programming process is slightly different since the external address counter and tristate buffers have to be enabled.

When the bootstrap program takes over for the HMOS devices, a reset signal (RST2) is produced by PB4, resetting the address counters to 0. The RAM data is then read via Port A into the controller's PROM and a clock pulse is produced by PB3, incrementing the address counter. The next data byte from RAM is burned in the PROM. The cycle continues until all the data has been transferred.

Once all the address locations in the PROM are programmed, the HMOS device makes PB1 low to indicate the completion of programming. The RST2 line, PB4, is pulsed which resets the address counters to 0. Also, PBO is made low, which turns off the 21 VDC at the V_{PP} input pin.

Now the 68705 is ready to check that it has successfully programmed its internal PROM. This is done by reading the data byte presented to the 68705 on Port A and comparing it to its internal PROM. The 68705 requests a new byte by pulsing PB3 as with the program cycle. If the internal PROM matches the data presented to it on Port A, a verified condition is signaled by making PB2 low. The PC then gives a message that programming has been successful and switches off power to the programmer ZIF socket.

OFF TO THE RACES

The development system has proven to be a powerful development tool for the Motorola 68HC705 microcontrollers. I'm now working on an integrated text editor and cross assembler so users can develop and debug 6805 code without seeing a DOS prompt.

As for the car computer.. .well, it survived the race, but the car didn't. Since that original trial, the car computer has been successfully tested in many cars with outstanding performance. Perhaps 1'11 describe the car computer in another article. ❑

Robert Priestley is an electronics engineer who provides technical support for a large communications company. He specializes in data communications and has interests in single-chip micros. He may be reached at oztec@ozemail.com.au.

SOURCE

The simulator package supports the 68HC705C8, J2, K1, C9, D9, El, P9, and B5, and the 68705P3, U3, and R3 microcontrollers. The programmer board supports the 68HC705 C4, C8, J2, and K1, and the 68705 P3, U3, and R3 microcontrollers.

Complete kit (software, manual, PCB, and components)	$195
Software, manual, PCB	$140
P9 programmer adapter board	$20
11 Adaptor boards for B5, P9, and El programming (supports ICS for J2, K1, P9, El, P3, U3, and R3)	$105

Note: All prices are in U.S. dollars and include shipping and handling.

Oztechnics Pty., Ltd.
Robert Priestley
P.O. Box 38
Illawong NSW
Australia
61-2-541-0310
Fax: 61-2-541-0734
E-mail: oztec@ozemail.com.au

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.

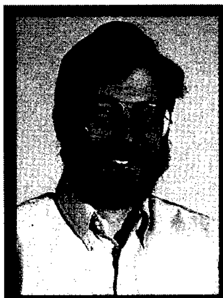
I R S

410 Very Useful
411 Moderately Useful
412 Not Useful

DEPARTMENTS

Ed Nisley

Journey to the Protected Land: Memory, Time, and I/O



Need your RAM checked? Ed's here to do it. Two simple protected-mode dynamic memory allocation routines slice up the storage into task data, stacks, and temporary buffers. He wraps up his check with a look at the clock.

back in the Bad Old Days, when microcomputers were fresh and new, a few precious kilobytes of memory were entirely enough for an operating system and perhaps an application program or two. Nowadays, 4 MB of system RAM is completely inadequate, entry-level systems sport 8 MB, and 32 MB looks positively mainstream. Ever wonder what all that memory is remembering?

This month, I'll start by reviewing the FFTS's RAM layout. Even though FFTS is simplified to the point of anorexia, it remains a 32-bit protected-mode operating system running on an Intel '386SX CPU. Quite unlike its distant commercial relatives, though, FFTS loads from a 32-KB disk file and runs happily in about 320 KB of RAM.

Most '386SX systems have a megabyte or three of memory between the FFTS kernel and the end of system RAM. I'll describe two simple PM dynamic-memory-allocation routines which slice that storage into task data, stacks, and temporary buffers. The issues are relevant for any operating system or application program, so pay attention.

Finally, I'll use the system board's RTC to produce a periodic tick, fire up an interrupt handler, and look at a peculiar timing. We have just enough

Listing l--The *FFTS kernel uses the first 320 KB* beyond *the 1-MB* address line, even *though the disk file holds less than 32 KB of code and* initialized data. *This fable shows where the various kernel segments begin. The two memory* allocators parcel *out* storage *between* BASE_MALLOC *and the end of RAM.*

```
BASE-LOAD       = 000100000h ; 1 MB load address

BASE_GDT        = 000010000h; GDT offset
BASE_IDT        = 000020000h ; IDT offset
BASE-STACK      = 000022000h ; stack offset
BASE_TSS        = 000030000h ; start of TSS array
BASE-DATA       = 000038000h ; kernel initialized data (max 32KB)
BASE_TASKDATA   = 000040000h ; task FAR data classes (max 64 K)
BASE_MALLOC     = 000050000h ; start of free space
```

space for a quick look at support routines for the little character LCD panel, watchdog timer, and DS2400 serial number on the FDB.

Think of it as knuckle cracking before we get down to really complex code!

DYNAMIC RAM TENSION

The constants in Listing 1 define the initial segment starting addresses required to get FFTS off the ground. Regardless of how you implement a PM program, you'll need these same segments or a functionally equivalent assortment. I picked convenient addresses with no attempt at wringing out the last byte of storage.

The show begins at 00100000, the l-MB line, where the PMLoader program places the contents of the FFTS.PMO disk file after booting from diskette. FFTS uses all of the installed RAM from that point upward without working around refresh buffers, BIOS ROMs, and the myriad other inconveniences found below 1 MB. We'll use the 640 KB of "conventional memory" below the line when we build Virtual-86 mode tasks.

The FFTS.PMO disk image occupies the first 64-KB block at 1 MB. The current FFTS.PMO file is a mere 32 KB: the code and data are only 23 KB long followed by 9 KB of padding. The Paradigm Locate program produces binary files in power-of-two sizes regardless of the actual program length.

The FFTS set-up code creates the system GDT in the next 64-KB block. Most FFTS descriptors cluster near the start of the GDT with the remainder spread thinly throughout the rest of the table. For example, the _Conform code segment's descriptor is at 4000 and the system call gates begin at 8000. You can compact your system's GDT by moving (or eliminating!) the conforming-code segment descriptor and reassigning the remaining gates to numerically smaller selectors.

I tucked the IDT and kernel stack into the next 64-KB block. All the unused IDT descriptors point to an unexpected interrupt handler, thus puffing the table to its maximum 2-KB (256 entries x 8 bytes) extent. The stack starts at the next 8-KB boundary to make its address easy on the eyes. However, now that we have multiple tasks with independent stacks, you can certainly trim the kernel stack to something more reasonable than EOOO bytes.

Next comes the 32-KB area reserved for TSS. Each TSS includes, in addition to the mandatory fields described in the Intel manuals, the task's LDT and an ASCII task name. An I/O permission bitmap at the end fills each TSS up to about 400 bytes. I plumped that to 5 12 bytes, making nice hex numbers out of successive TSS addresses. If you need more than 64 tasks, you can shrink the TSS entries or expand their RAM allotment.

Finally, we find some data. The start-up code copies the kernel's DGROUP class from the disk image into a separate 32-KB area and creates the GDT_DATA descriptor over it. That segment holds all the NEAR data, both initialized and uninitialized, used by the kernel's SMALL model code. Although the data may occupy up to 32 KB, the descriptor covers only the data actually present, thus preventing off-the-end access errors.

Each FFTS task has its own initialized data segment that is not shared with any other segment. The real-mode tools we're using limit each task data segment to 64 KB, but PM Loader's 64-KB file-size limit cramps our style long before that. The FFTS start-up code copies all of the task data segments from the disk image into a single 64-KB area. The task creation code then subdivides that by creating separate LDT_DATA descriptors for each task's portion.

You could eliminate that block copy by simply aiming the LDT_DATA descriptors at the initialized task data in the disk image. At some point, I'll put FFTS in the FDB's NVRAM to produce a boot-to-PM computer, making a RAM copy absolutely mandatory.

Conversely, the constant data segment, _protconst, need not be copied because it's accessed through a read-only data descriptor. An EPROM or NVRAM home for this segment is a natural!

The start-up code creates a final data segment covering the storage from just beyond the kernel to the end

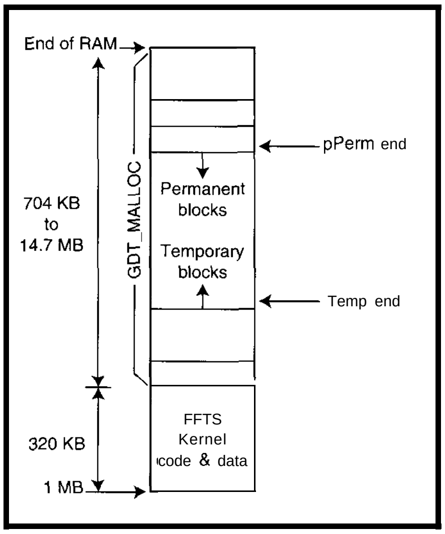


Figure 1 —*The dynamic memory allocators subdivide the memory between fhe FFTS kernel and fhe end of RAM. Permanenf memory blocks start at fhe highesf address, temporary blocks start at the lowest, and a single giant data segment covering the entire expanse gives the kernel access to the memory block headers preceding each* block. *Two pointers track the extent* of *each group.*

Listing 2—*This 64-byte header precedes each dynamically allocated memory block. The task receiving this block cannot access the header because the block descriptor covers only the data area. The header is only accessible to kernel routines using the* `GDT_MALLOC` *descriptor.*

```
;--- values in the Status field
BLK_UNUSED   =  0            ; memory block was never used
BLK_FREED    =  1            ; used, later freed
BLK_ALLOCATED = 2            ; currently in use
BLK_PERMANENT = 3            ; permanently allocated
BLK_SYSTEM    = 4            ; special system use

MAX_DESCRIPTION     = 35     ; max length of description field
                             ; to make an even 64-byte header

             STRUC  MEMBLKHEAD
TaskID       DW     ?        ; task ID that created block
Selector     DW     ?        ; selector for data block
BaseAddr     DD     ?        ; data block starting address
DataSize     DD     ?        ; data block size in bytes
AllocSize    DD     ?        ; allocated block size in bytes
Status       DD     ?        ; block status
UseCounter   DD     ?        ; number of times block is used
OwnerID      DD     ?        ; ID value supplied by owner
Description  DB     MAX_DESCRIPTION DUP (?); owner's description
             DB     ?           ... terminator
             ENDS   MEMBLKHEAD;

MBH_SIZE     =      SIZE MEMBLKHEAD

MBH_PTR      EQU    <(MEMBLKHEAD PTR ES:EDI)>
```

of the system's RAM. It sets the descriptor's G bit because the segment may span up to 14.7 MB of storage in a full-up '386SX. A '386DX system can have up to 64 MB of RAM before the BIOS "Get Extended Memory Size" call runs out of bits in AX. In any event, this is a *big* segment.

You can probably guess the punch line from the descriptor's name: GDT_MALLOC. Yes, FFTS treats that expanse of RAM as a heap of storage (in the computer science sense of the word, not the old-laundry meaning). Because FFTS runs in protected mode, it can do things both differently and better than your average real-mode memory allocator.

PARTITIONING THE PROBLEM

We first encountered dynamic memory allocation in ***INK 55*** while creating FFTS tasks. The stack and uninitialized data segments for each task are ideal candidates for dynamic allocation because neither contains initial values and both can be created on the fly. The task-creation code carved out two appropriately sized chunks of unused memory, filled in a pair of descriptors, plunked the corresponding selectors into the task's TSS, cleared the storage, and that was that.

An alternative method, hard-coding the segment addresses and descriptors into the FFTS kernel, is also workable. You might prefer static allocation when you have only a few tasks with very well-defined storage requirements or no tasks at all. In fact, that's why it's taken so long for me to get around to discussing dynamic allocation: you can get quite a lot of code running without it!

The FFTS kernel creates tasks that run forever (or until the next reset) and, thus, must have their stack and data segments available at all times. The tasks may also request and release blocks of storage as they run, which means that the allocator must recycle unused storage. The FFTS allocators take advantage of the difference between permanent and temporary memory blocks.

Figure 1 presents a roughly scaled drawing of the FFTS storage layout. The GDT_MALLOC descriptor gives access to the entire block of RAM beyond the kernel. Temporary blocks form a stalagmite, permanent blocks are a stalactite, and should the two fuse into a column, you need more RAM!

MemAllocPerm parcels out permanent blocks starting at the highest addresses in GDT_MALLOC. Because allocations occur in first-come, first-served order, the pPermEnd pointer indicates both the start of the most recent block and the entire extent of permanently allocated storage. Allocating a new block is a simple matter of adjusting pPermEnd downward by the new size.

MemAlloc, in contrast, searches through all of the existing temporary blocks to find space for a new block. It must find a free block of the right size or, if all the blocks are in use, carve a new block out of unused RAM beyond pTempEnd. Because blocks are freed when they're no longer needed, pTempEnd marks the beginning of the never used part of GDT_MALLOC, rather than the most recent block.

Many storage-allocation systems create both permanent and temporary blocks from the same pool. We are free to use two allocators because FFTS can take advantage of them. It need not be particularly compatible with dusty-deck programs dating back to the dawn of computing history. Analyze your program's requirements, and lay out your storage accordingly!

FLEETING MEMORIES

Program errors involving dynamically allocated storage are fiendishly difficult to track down. The FFTS allocators reduce this problem by creating a header for each new block containing all manner of interesting and useful information. The 64-byte header precedes its block of storage, hence the name. Listing 2 shows the fields in each header.

Real-mode memory allocators also use block headers and, in fact, memory errors in early PC spawned an entire cottage industry. An errant real-mode task, however, may destroy its own storage and memory-block headers, storage allocated to other tasks, operating-system data, and anything else within reach of a wild pointer. That's impossible in protected mode because the storage allocators create a descriptor covering only the requested block. The task cannot overwrite a block header (or anything else) because it does not have the appropriate descriptor.

```
Find empty descriptor from 0100 in 0008...
 0100 is 00000000 00000000 at 0100
MemAllocTemp 0100 in 0008 Size=00001000 Acc=92 Attr=40 TSS=1010
 ID=00001010 Fill=00000000 [ConfFormat output buffer]
Creating descriptor 0100 in 0008:
 Base=00000040 Size=00001 000 Acc=92 Attr=40
```

Figure 2—*The memory-management routines can display a* detailed *trace* of their activities. This record shows the sequence of events during a single memory allocation. The first *two* lines show *the* search *for an empty GDT descriptor:*

The fact that each memory block drags along a 64-byte header tells you that the FFTS memory allocators work best with larger blocks. In fact, the allocators round each request up to the next multiple of 64 bytes, meaning that a request for 4 bytes actually soaks up 128 bytes: 64 bytes for the header, 4 bytes of data, and 60 bytes of padding. The descriptor returned to the task covers only the 4 data bytes to detect an off-the-end access.

Rounding the request size solves two problems: it makes the addresses easier to read and helps reduce heap fragmentation. The latter is certainly the most important justification because MemAllocTemp searches for a free block with exactly the same size as the new request. Forcing all requests to a reasonable size helps ensure a ready supply of reusable blocks.

When the heap becomes badly fragmented with odd-sized blocks, the allocator cannot find a block that

Listing 3—*MemAlloc* Temp allocates a temporary memory block and returns a selector. *It* scans *the* memory-block header chain *starting* with *the first* temporary *block until if finds a free block of the same size or runs off the end of the chain* info unallocated storage. *In* either case, *it* fills in a *new* header and creates *a descriptor covering the data part of the block. A practical, heavy-duty allocator has many more features. This* one was optimized for simplicity!

```
      PROC  MemAllocTemp
      ARG   Selector:DWORD,TableAlias:DWORD,   \
            Size:DWORD,Access:DWORD,Attr:DWORD, \
            DescSeg:DWORD,pDesc:DWORD,OwnerID:DWORD,Fill:DWORD
      LOCAL AllocSize:DWORD
      USES  EAX,ECX,EDX,EDI,ESI,DS,ES

      MOV   EDX,GDT_DATA
      MOV   DS,DX

<<< trace code omitted >>>
;--- if the block is zero-length we bail out with a zero selector
      MOV   EAX,[Size]            ; round to next grain size
      ADD   EAX,MALLOC_TEMPGRAIN-1
      AND   EAX,NOT (MALLOC_TEMPGRAIN-1)
      MOV   [AllocSize],EAX

      JZ    @@NoAlloc             ; if zero, do not allocate block

;--- search for block big enough to hold allocation request
      MOV   ES,[pTempEnd.Seg]     ; aim at first temp block
      XOR   EDI,EDI

@Search:
      CMP   [MBH_PTR.Status],BLK_FREED; check block status

      JA    @@Skip                ; in use, skip it
      JB    @Carve                ; unused, carve it up

      CMP   [MBH_PTR.AllocSize],EAX    ; freed, can we use it?
      JE    @@DoAlloc             ; exact size match. yes!
```

(continued)

Listing 3—*continued*

```
@Skip:
      ADD   EDI,[MBH_PTR.AllocSize]    ; skip to next block
      ADD   EDI,MBH_SIZE
      JMP   @Search

;--- no block that fits, carve a chunk from unallocated space at
     the end first, make sure the whole block fits below the
     permanent area ED1 points to the start of the new header
@@Carve:
   MOV      EAX,EDI              ; figure end of new block
   ADD      EAX,[AllocSize]      ; . . . with data block
   ADD      EAX,MBH_SIZE         ; . . . and block header
   CMP      EAX,[pPermEnd.Off]   ; must be below lowest perm block
   JB       @@NewBlock           ; strictly below is OK

   TrcIf    TrcMemAlloc
   CallSys CGT_SER_SENDSTR,GDT_CONST,<OFFSET cMsg_AT2>
   TrcEndIf

   CALL     MemKillDescriptor,[Selector],[TableAlias]

@@NoAlloc:
   XOR      EAX,EAX                    ; error or zero-length block
   JMP      @Done

@@NewBlock:
   MOV      [pTempEnd.Off],EAX         ; record new high-water mark

;--- OK to allocate: fill in the block header
     ED1 points to the block header
@@DoAlloc:
   STR      DX
   MOV      [MBH_PTR.TaskID],DX
   MOV      EAX,[Selector]
   MOV      [MBH_PTR.Selector],AX
   LEA      EDX,[ES:EDI + MBH_SIZE]    ; block starting offset
   MOV      [MBH_PTR.BaseAddr],EDX
   MOV      EAX,[Size]
   MOV      [MBH_PTR.DataSize],EAX
   MOV      EAX,[AllocSize]
   MOV      [MBH_PTR.AllocSize],EAX
   MOV      [MBH_PTR.Status],BLK_ALLOCATED
   INC      [MBH_PTR.UseCounter]

   MOV      EAX,[OwnerID]
   MOV      [MBH_PTR.OwnerID],EAX
   LEA      EAX,[MBH_PTR.Description]

   CALL     StrfNCopy,GDT_MALLOC,EAX,MAX_DESCRIPTION, \
            [DescSeg],[pDesc]

   CALL     MemSetDescriptor,[Selector],[TableAlias], \
            EDX,[Size],[Access],[Attr]

;--- fill  the entire allocated block with a marker value
   MOV      EDI,EDX                    ; aim ES:EDI at block
   MOV      ECX,[AllocSize]            ; fill the whole block
   SHR      ECX,2                      ; . . .  as dwords
   MOV      EAX,[Fill]                 ; set up caller's fill value
   REP STOS CDWORD PTR ES:EDI]         ; poof!

   MOV      EAX,[Selector]             ; return selector

@Done:
   RET

   ENDP     MemAllocTemp
```

matches a request. A heavy-duty heap manager goes through a garbage-collection cycle, combines small adjacent free blocks, splits large blocks, and generally does whatever is necessary to find a suitable block. The FFTS allocator simply gives up in disgust, spits out an error message, and returns a zero selector that causes a protection error when it is used later on.

PM descriptors give you a powerful heap-management tool because tasks access memory blocks using selectors rather than actual storage addresses. You can move the blocks around in memory by changing the base address field in the block descriptors; the tasks and selectors aren't affected. This is essentially impossible in real mode: witness the memory gyrations required in Windows prior to Version 3.1.

With all that as background, MemAllocTemp appears in Listing 3. Unlike malloc() in the C library, this routine expects a variety of information detailing who requested the memory, the block size required, what the block will be used for, and how it will be accessed. In a few months, we'll encyst this procedure in a wrapper that supplies most of the header values automatically. For now, all the control knobs stick out in plain view.

The trace output directed to the serial port during a single memory allocation appears in Figure 2. In this example, ConfFormat called MemAllocTemp to create a 4-KB buffer for its output string. The allocator searched for an empty GDT descriptor, located and initialized the block, then filled in the new descriptor's fields. Note the descriptor's base address of 00000040, just after its 64-byte header, shows that it is the first block in GDT_MALLOC.

MemAllocPerm is a similar chunk of code, except that it need not search for a vacant block. The BBS code this month also includes MemFree and a clutter of helper routines, call gates, and suchlike. The code is reasonably straightforward once you get used to the notions of a selector for every descriptor, a descriptor for every block, and careful attention to never touching unallocated storage.

Listing 4—The FFTS kernel measures time using the RTC's Update and Periodic Interrupts. The Update Interrupt occurs when the RTC has just finished updating the current time; this tells the interrupt handler to reset its fractional-second counter. Periodic Interrupts occur 64 times per second, each one adding 15625 counts to the fractional-second counter. Photo 1 shows the timing relation between the two interrupts.

```
         PROC TmrRTCHandler FAR
         USES EAX,EDX,DS

         MOV   EDX,SYNC_ADDR      ; show a tick
         IN    AL,DX
         OR    AL,40h
         OUT   DX,AL

         MOV   EAX,GDT_DATA       ; get addressability to our data
         MOV   DS,AX

         MOV   AL.RTC_REGC        ; read Reg C to clear interrupt
         OUT   RTC_ADDR.AL
         IN    AL,RTC_DATA        ; PF or UF may be set, now cleared

         TEST  AL,MASK RegC_PF;   periodic interrupt?
         JZ    @@NoTick
         INC   [TickCounter]      ; yes, record another tick
@@NoTick:

         TEST  AL,MASK RegC_UF;   end of update?
         JZ    @@MidSec           ; no, keep counting
         IN    AL,DX              ;  yes, mark it
         OR    AL,20h
         OUT   DX,AL
         AND   AL,NOT 20h
         OUT   DX,AL

         MOV   [FracSec],0        ; and reset to whole second
         MOV   [TickSize],FIRST_TICK  ; next tick is short
@@MidSec:
         MOV   EAX,[TickSize]     ; increment microsecond counter
         ADD   [FracSec],EAX
         MOV   [TickSize],TICK_US      ; remaining ticks are OK

         MOV   AL,NS_EOI          ; now reset the 8259 ISRs
         OUT   I8259B,AL          ;EOI secondary controller
         OUT   I8259A,AL          ;EOI primary controller

         MOV   EDX,SYNC_ADDR      ; remove the tick
         IN    AL,DX
         AND   AL,NOT 40h
         OUT   DX,AL

         POP   DS                 ;*** no automatic POPs!
         POP   EDX
         POP   EAX

         IRET                     ; return to interrupted code
```

The FFTS memory allocation routines are about as simple as they can be. Your applications may need more memory-management firepower. There are many computer science books out there to help you take aim before firing. In any event, you can have a good deal of fun complexifying the FFTS code beyond recognition.

TICKING TODAY'S TIME

Judging by the profusion of clock-calendar chips available nowadays, everybody expects computer systems to know what time it is. Jeff's RTC review in *INK 52* covered a dozen or so, and I'm sure more have appeared on the market since then. Fortunately for us, the PC Compatibility Barnacles ensure each system board sports a full-function clock-calendar compatible with the venerable MC146818A.

Well, mostly compatible-if you don't try anything too peculiar. Newer PC chips include a huge assortment of modes, special registers, additional storage, and what have you lying in wait for the innocent user who missets a configuration bit. I strongly recommend that you don't fiddle with bits labeled "don't care" or "not implemented" and use the default settings for all bits you don't need. For sure, changing those bits will do something evil to somebody's silicon.

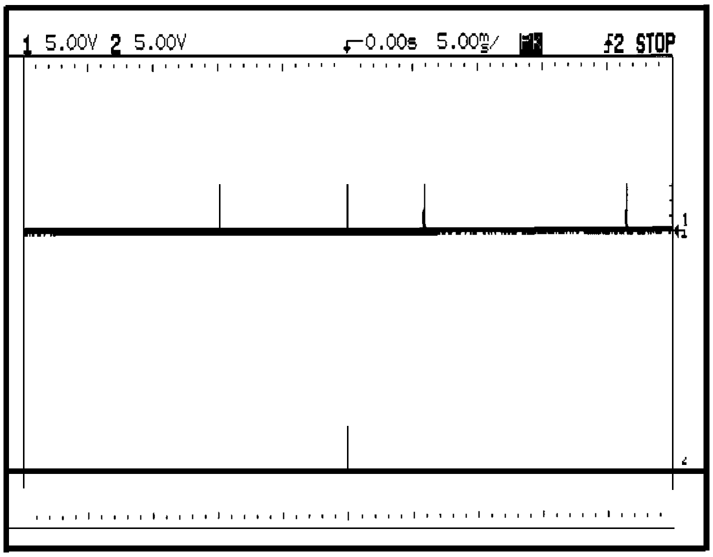


Photo 1—The RJC interrupt handler produces two parallel port blips that trace its action. The Periodic Interrupt pulses occur 64 times per second. The Update Interrupt pulse in the center of both traces occurs once per second, about ⅝ of the way between the two PIs bracketing fhe update. The code in Listing 4 adjusts the fractional-second counter to compensate for the difference.

FFTS uses two plain-vanilla RTC interrupts: the Periodic Interrupt [PI] at 64 ticks per second and the Update Interrupt (UI) occurring once per second. The Compatibility Barnacles dictate that the RTC must use IRQ8 on the system board's secondary (slave) 8259. Our set-up code maps that interrupt into the usual Int 70. The interrupt hardware and vector setup is familiar from previous columns; I won't devote any space to it here.

I picked 64 Hz because it makes a nice preemptive task-switcher timebase: a '386SX task can get a reasonable amount of work done in 15.625 ms. No, the task switcher hasn't mutated while you weren't watching. I'm just laying some groundwork should we ever do such a thing. Feel free to adjust the rate to suit your own purposes.

The RTC maintains the current time and date to the second, relieving us of the need to convert ticks into wall clock time. Many projects require time resolution better than one second, though, and it's easy enough to synchronize a fractional-second counter with the RTC interrupts.

Listing 4 shows how that works. FracSec contains the number of microseconds since the last UI. Each

UI clears FracSec and each PI adds TickSize to FracSec. Contraryto what you might expect, TickSize is a variable rather than a constant.

Photo 1 shows why TickSize must have two values. The UI occurs about ⅝ of a cycle (9.766 ms) after a PI. The next PI adds 5.859 ms to FracSec, while the remaining 63 interrupts each add 15.625 ms. Those tidy rational numbers are within a few microseconds of the actual values and, as the error doesn't accumulate for more than a second, are close enough for our purposes.

Listing 5-A single 16-bit output port on the FDB controls the character LCD panel, watchdog timer, DS2400 serial number, and fhe nonvolatile RAM write-enable line. The code for all four devices must maintain a consistent output value, which is easy enough if they read and update a common variable in RAM. This code shows the port bit definitions and a routine that toggles the watchdog bit The code must disable interrupts whenever the output port does not match the FDBControls variable.

```
;--- define the output port bits
        RECORD   FDB_CTLS{
          FDB_Unused_HighOut:16,
          FDB_TLCD_DisData:1=1,
          FDB_TLCD_RegSel:1,
          FDB_TLCD_RdWr:1,
          FDB_TLCD_Enable:1,
          FDB_Unused_BOut:1,
          FDB_SerNumOut:1=1,
          FDB_WriteEn:1,
          FDB_WatchDog:1,
          FDB_TLCD_Data:8


<<< other bits & variables omitted >>>

        PROC     UtilToggleDog
        USES     EAX,EDX

        PUSHF
        CL1

        MOV      EAX,[FDBControls]
        XOR      EAX,MASK FDB_WatchDog
        MOV      EDX,CTLS_ADDR
        OUT      DX,AX
        MOV      [FDBControls],EAX

        POPF
        RET

        ENDP     UtilToggleDog
```

The TmrGetTime function, which I don't have space to present in detail, extracts the current hour, minute, and second from the RTC registers. It then converts FracSec from microseconds into hundredths of a second. The results are returned in packed BCD in EAX as HHMMSSFF. That 32-bit register is just big enough for everything we need!

Despite the fact that FracSec counts microseconds, it takes on only 64 different values during each second. FracSec is zero immediately after the UI, steps to 5859 on the first PI, 21484 on the next, and is 990234 after the sixty-fourth PI. Don't call TmrGet-Time expecting to see a uniform distribution of fractional seconds!

The timer interrupt handler also increments TickCounter during each PI. The TmrGetMS function (also not shown here) converts TickCounter into milliseconds and returns the result in EAX. The 32-bit value wraps every 49.7 days, which seems reasonable under the circumstances.

Most RTC crystals are at least a little off their exact 32.768-kHz spec. A 0.01% error (a mere ±3 ns per cycle) skews the time by about four minutes per month. If you find that intolerable, here's an extra-credit project: measure your RTC crystal's error and write a smidgen of code to adjust the clock once a day, on every reset, or when the error exceeds one second.

Need some hints? The RTC gives you the unadjusted time; multiply that by the appropriate Fudge Factor to come up with time error. The frequency error may change when the RTC runs from its battery. The RTC measures time by whole seconds—should you remember fractional seconds of error between tweaks?

One final hint: backing up or advancing the clock by as little as one second can change all the RTC registers. Perhaps you should sort of sleaze up on the right time during the next few seconds or minutes?

SEPARATE HARDWARE, COMMON BITS

In *INK 39,* I described the FDB's character LCD, DS2400 serial number, and watchdog timer. The code that month used good old 16-bit, real-mode Micro-C. It's a year and a half later, we're in 32-bit protected mode, and that makes less difference than you'd imagine.

The key problem remains that the three devices and the nonvolatile RAM write-enable line all share a 16-bit write-only output port. Every routine that changes the port must know the current value of the other bits. The only practical way to ensure that is by storing the value in RAM each time you write the port and using that stored value for each change.

Listing 5 illustrates two essential tricks. Each chunk of code that changes the output port must update its bits in the FDBControls variable to let everyone else know their current value. The code must disable interrupts whenever the port doesn't match the variable to prevent anyone else from using the wrong bits. Because each routine changes only the bits for its output device, the other bits remain stable regardless of how often the port is written.

Although I don't have room for more of the source code this month, the entire collection is spinning

Acronyms

CPL	Current Privilege Level
DPL	Descriptor Privilege Level
EOI	End Of Interrupt (command)
FDB	Firmware Development Board
FFTS	Firmware Furnace Task Switcher
GDT	Global Descriptor Table
G bit	Granularity bit (in a PM descriptor)
IDT	Interrupt Descriptor Table
IF	Interrupt Flag
IOPL	I/O Privilege Level
LDT	Local Descriptor Table
NT	Nested Task
P bit	Present bit (in a PM descriptor)
RF	Resume Flag
TF	Trap Flag
TR	Task Register
TSS	Task State Segment

around on the BBS. All the low-level code for these gadgets resides in UTILITY.ASM to keep it close to the FDBControls definition. The DS2400 and watchdog are simple enough that there isn't any high-level code!

I turned the little character LCD into a console output device using code cribbed from the graphic LCD and video interfaces described in *INK 53*. Fans of reusable code should be appalled: I simply copied the source, twiddled it to reflect the differences, andpoppeditintoTEXTLCD.ASM.If your system lacks a character LCD, you can discard the code without paying an over-generalized code penalty. Fair enough?

RELEASE NOTES

Demo Task 1 can still trigger a protection exception on demand. Demo Task 2 requests the current time-of-day and millisecond tick counter using the new RTC code and displays them on the VGA. Demo Task 3 displays a running count on all three displays: VGA, graphic LCD, and the little character LCD. The RTC generates 64 interrupts per second to update the millisecond timer.

All three taskettes use the new console-output routines and exercise the memory-allocation code. We'll see more on memory-allocation code later, making this month's coverage a brief introduction and sanity check.

Frank Van Gilluwe's *The Undocumented PC* (Addison Wesley, ISBN 0-201-62277-7) is a category killer for PC-hardware junkies. Read past the editing gaffes (starting in the preface's first sentence, ouch) to the hard-core technical information. Finally, I can retire my dog-eared *IBM Tech Reference* manuals!

Steve Maguire's *Writing Solid Code* (Microsoft Press, ISBN l-55615 55 l-4) includes a good discussion of techniques to keep you out of trouble, along with sample code to get you out once you're in. An augmented memory manager for your heap is just one of the lesser topics.

Michael Abrash's *Zen of Code Optimization* (Coriolis Group Books, ISBN l-883577-03-9) mentions protected mode only in passing. It contains updated chapters from his now-out-of-print *Zen of Assembly Language* master-work along with excerpts from his "Pushing the Envelope" columns in PC *Techniques*. FFTS isn't concerned with raw performance. If you are, this book belongs on your desk.

Remember that some folks using your product may be blind. Contact Wayne Thompson at the Kentucky Department for the Blind for a proposed standard interface that enables a blind user to "see" what's on a HD44780 LCD panel. His numbers are (502) 564-4754 and (502) 564-3976 (fax). The test code this month updates the LCD far too often, but a real application would be more polite.

Next month, we begin using the keyboard in protected mode. Prepare for a descent to the lowest levels of grubbiness! ❑

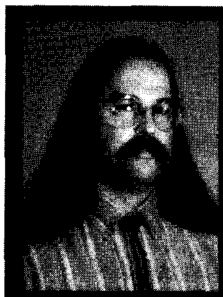
Ed Nisley, as Nisley Micro Engineering, makes small computers do amazing things. He's also a member of Circuit Cellar INK's engineering staff. You may reach him at ed.nisley@circellar.com. or 74065.1363@compuserve.com.

IRS

413 Very Useful
414 Moderately Useful
415 Not Useful

Time in a Can

Just Add 1 Bit of I/O



It's almost time to open the cooler and chug back those cans. This Bud's real cool. It offers a combination of elapsed time, enabled counts, ID serial number, and NVRAM.

FROM THE BENCH

Jeff Bachiochi

No, I didn't find it at the bottom of my cooler, but you're close. This press release, dated November 19, **1991,** came from beneath a rather large pile in the corner of my office. It turned up again (as it has done for the past four years) in the annual office shake down.

Once a year, I go through every item located within the confines of my home away from home and determine what stays and what goes. The fact that this release made it through numerous efforts to clean house must count for something. Either that, or I'm highly susceptible to a clever advertising gimmick?

TIME IN A CAN (TIC)

It's about the size of a can of beans (see Photo 1), but no can opener is needed because of its pull-tab top. The can's label touts the general directions: "Hour Meter Only-just connect to V_{CC} (+5 V) and ground. Full Features—connect to single I/O signal plus ground for real-time clock, calendar, alarm, stop watch, interval timer, counter, log book, serial number, lock, and retrofit."

There is a strange rattling sound, which reverberates from the can, as if the contents were all dried up. The ingredients are listed on the label as silicon, quartz, lithium, and stainless steel. Ooh, sounds yummy. Luckily, there is a phone number at the bottom of the label just under the words "For Questions and Comments Call," which connects you to Dallas Semiconductor.

I pop off the lid and out falls a small steel can about the size and thickness of two nickels. Close inspection shows the can's lid is insulated from its body like a coin-battery cell. A coin battery-cell holder can be used to both secure the TIC and make electrical contact with it.

Let's look inside this device.

SILICON, QUARTZ, LITHIUM, AND STAINLESS STEEL

Figure 1 is a block diagram of the DS 1994 NVRAM and time touch-memory device. The ROM area of the DS 1994 contains a unique laser-trimmed serial number (see FTB, ***INK* 24).** The total ROM ID number has

Photo 1—*Talk about off the shelf! Check the silicon section of your local distributor.*

three parts: a I-byte family code {06H for the DS1994}, a 6-byte unique serial number, and a l-byte CRC. This information forms the device's unique address, which is used for talking to multiple devices connected in parallel to the same I/O pin. When communicating though a single I/O pin, devices not matching the current address disregard communication.

Following the ROM area is an NVRAM area set up in pages of 32 bytes each. The first page is called the scratchpad and is used to verify data prior to transferring it to one of the higher pages used as permanent storage. As Figure 2 illustrates, the DS1994 has 16 pages of permanent NVRAM storage (4 Kb).

The last page of storage is where most of the action takes place. The first 30 bytes of this page are used as follows: device status register (one byte), device control register (one byte), real-time counter registers (five bytes), interval-time counter registers (five bytes), cycle-counter registers (four bytes), real-time counter-alarm registers (five bytes), interval time counter-alarm registers (five bytes), and cycle counter-alarm registers (four bytes)

To clearly understand the intended applications, let's further investigate each register group beginning with the counters.

REAL-TIME COUNTER

Time here is represented in a complete binary fashion. A clock ticking once each 1/256 of a second increments a five-byte counter. This counter rolls over once every 136 years. The initial count can be set to zero (indicating this instant) or to a count equaling the number passed since a specific time and date.

The most common starting point selected is January 1, 1970, 00:00:00. I'll bet most of you have seen this date and time before. It is the default date and time for PCs which have not been set or have lost their configuration.

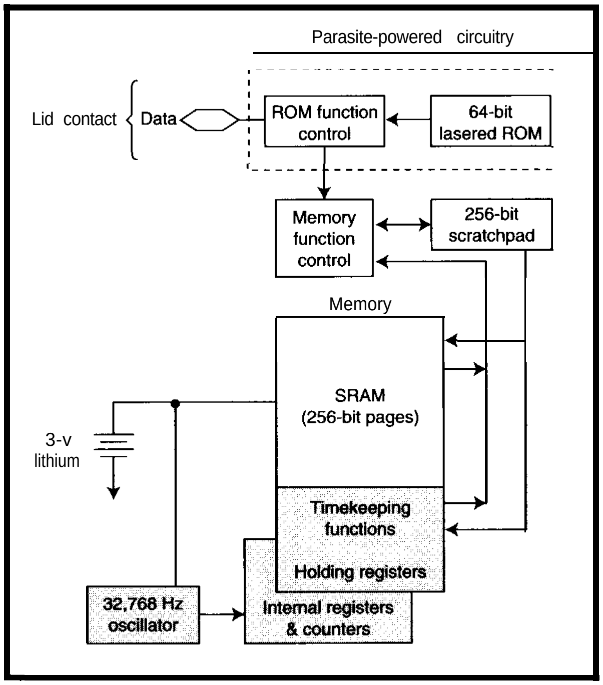


Figure 1—*Dallas Semiconductor's DS1994 includes both memory and timekeeping functions. The shaded areas show fhe time functions.*

Although it is up to the user to convert this count into an actual time and date, the algorithm is fairly simple.

INTERVAL TIME COUNTER

The second counter runs off the same clock as the RTC above, except it is gated. The gating or enabling of the counter comes from two sources depending on the bits in the device control register. In the manual mode, the user starts and stops the Interval Time Counter (ITC) through software commands. In the automatic mode, the ITC counts as long as a logic high (usually V,,) is applied to the DS1994. Thus, the counter is controlled by the I/O pin.

CYCLE-COUNTER REGISTER

The third counter monitors the DS1994 I/O pin and keeps a total of the number of power cycles the device goes through (not counting communication). This value can be helpful in setting maintenance schedules. For example, the ITC divided by the Cycle Counter indicates the average time on per cycle.

ALARM REGISTERS

Each of the counter registers has an alarm register associated with it. These alarm registers are the same size as their corresponding counter registers.

When set, the alarm registers produce a high flag in the device status register when a match occurs between an alarm register and its associated counter register. There is one flag for each of the counter registers, a Real-Time Flag (RTF), an Interval-Counter Flag (ICF), and a Cycle-Counter Flag (CCF).

DEVICE STATUS REGISTER

Alarm flags are located in the device status register. These flags are read-only bits, reset to zero after each device status register read. Three additional bits, one for each alarm, are also located in the device status register.

These read/write bits enable and disable interrupt reporting of the alarm flags. Since the DS1994 requires power to answer a request for NV-stored data, access should not be polled to watch for the alarm flags. Instead, enabling the alarm interrupt lets the DS1994 signal the user by pulling the I/O line down.

DEVICE CONTROL REGISTER

This register is the most important to the user for defining the modes in which the DS1994 operates. The DSEL bit chooses between two delay times (a short 4 ms or long 123 ms), which the interval and cycle counters use to determine whether the I/O pin has no activity (i.e., no actual communication) and should be considered at a constant high or low.

The AUTO/MAN bit selects the gating mode for the interval counter register. When this is in manual mode, the STOP/START bit disables or enables the interval counter register. The OSC bit turns the internal 32-kHz RTC/ITC timebase on or off. While this can be used to save internal lithium energy, the RTC and ITC cannot increment the count with this bit disabled.

The last four bits enable functions which cannot be cleared once they are set. These allow the device to be used as an expiration alarm for any or all of the counter functions. There is a separate bit to enable this write-protection function for any of the three counters: Write-Protect Real (WPR), Write-Protect Interval (WPI), and Write-Protect Cycle (WPC). Once any of these are set, the associated registers cannot be changed ever again.

In addition, a fourth bit, RO, can disable the whole NVRAM area whenever any of the write-protected alarms go off. At this point, only the ROM ID number can be read.

These protective features are radical in that the device cannot be used again. However, sometimes you want to provide this kind of protection in your product. And, if you don't, you won't have to worry about inadvertently setting these bits because they must be copied from the scratchpad area three consecutive times. Any changes to scratchpad between copies invalidates the transfer (i.e., clears the protection bits).

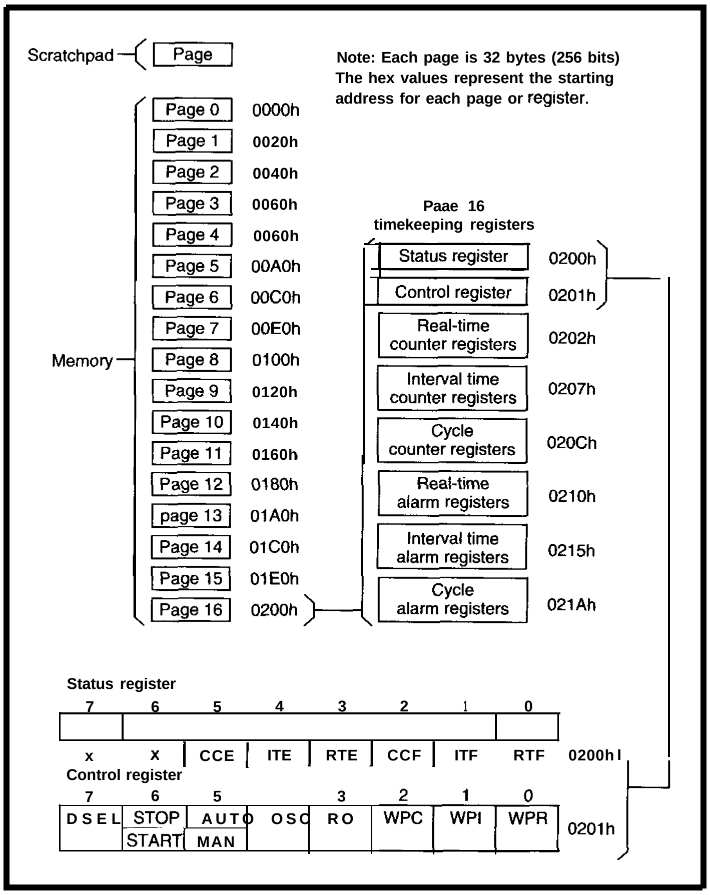


Figure 2—*The DS1994* has *16 pages of memory and27 timekeeping registers.*

COMMUNICATION PROTOCOL

All Dallas one-wire devices follow the Open System Interconnection (OSI) reference model of the International Standards Organization (ISO). This protocol has seven layers starting with the physical layer (defining electrical and timing characteristics) and ending with the application layer (the user-written program). Each layer uses the tools available from the previous layers to create new operations. In other words, physical timing is used to acquire bit passing. This promotes networking to transfer data by means of function calls in your application.

A typical command sequence consists of three phases. Each of these phases is initiated by the master (from your output bit to the touch memory) and answered by the DS1994 (from the touch memory to your input bit).

The first phase is a Reset Pulse which is answered with a Presence Pulse. The second is a ROM Command, which offers addressing information and may or may not need a reply depending on the command. The third phase is a Memory Function Command, which moves data to or from the touch memory. It also may or may not require a reply depending on the command.

Commands and data are sent least-significant bit first. Commands and data bits received by the DS1994 are reconstructed into bytes and stored in ascending addresses. Conversely, the user is responsible for receiving the data from the touch memory and rebuilding it.

How does the data transfer happen on a one-wire system? A one-wire connection assumes your hardware has at least one bit capable of bidirectional I/O. If not, two bits-one input and one output-can be used (see Figure 3).

COMMUNICATION TIMING

A reset pulse consists of lowering the output line to a logic low for a minimum of 480 µs. After raising it back to a logic high, you start monitoring the input. The DS1994 waits 15-60 µs before pulling the line to a logic low for 60-240 µs and releasing it. You must look for the presence pulse as an acknowledgment to the reset pulse.

Note that if phase two is not entered (i.e., the I/O is left in the logic high state and you continue to monitor the line), the touch device can signal an interrupt by pulling the I/O line to logic low for 960-3840 µs.

The remainder of the timing falls into three time slots: write a zero, write a one, and read data. To write a zero bit, drop the output for a minimum of 60-120 µs. To write a one bit, drop the output for at least 1 µs, but

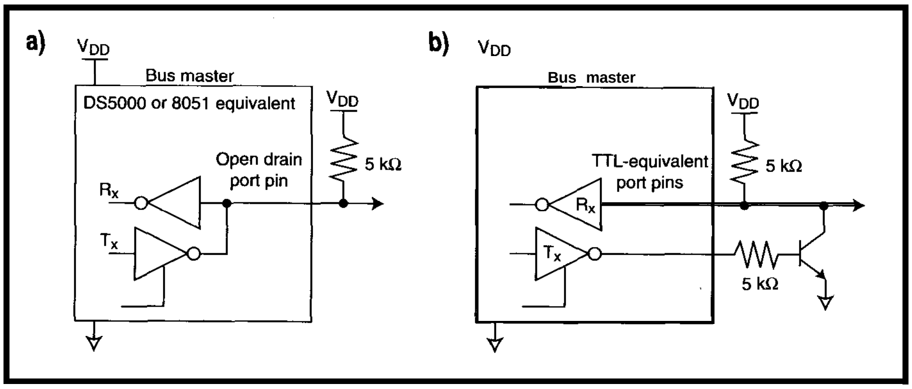


Figure 3—*The I/O port* connections *are for a) a* bidirectional *bit* and *b) two* unidirectional *bits.*

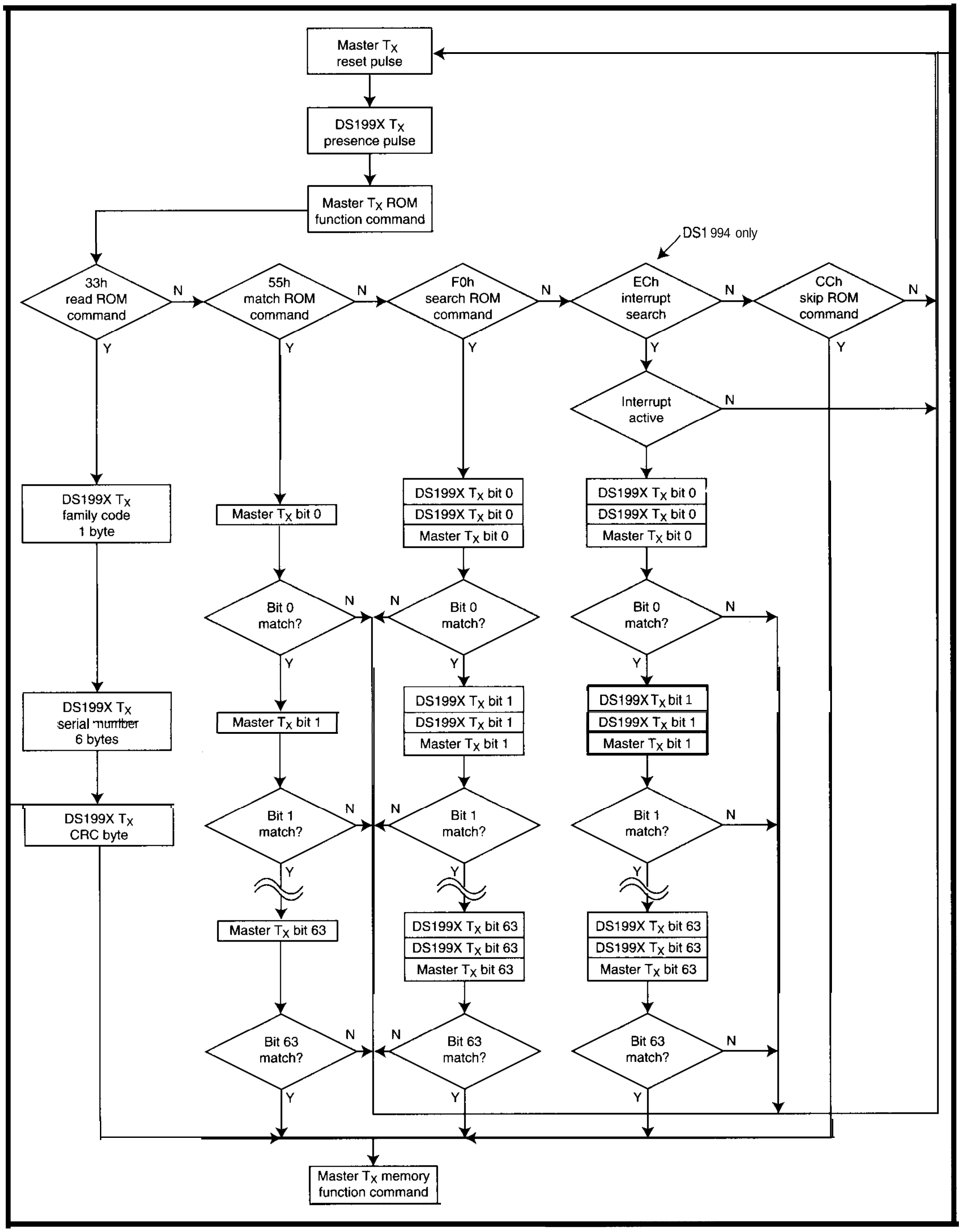


Figure 4-The first phase in the one-wire communication protocol is presence detection while the second phase includes ROM function commands.

less than 15 µs, and return it high for 60-120 us. To read a data bit, drop the output for at least 1 us, but less than 15 us, then return it high. If you monitor the line, it will remain high or be held low for a minimum of 60 us.

The following ROM command-byte sequences use the read/write timing above. (You can follow the ROM commands through the flowchart on Figure 4.) The Read-ROM command returns the family code, ID number, and CRC, but should only be used when the touch memory is the only device online since all devices answer.

The Match-ROM command sends out a family code, ID number, and CRC instructing only the device which matches the address to respond to the phase-three Memory-Function command.

The Skip-ROM command is the same as a Match-ROM command, except the family code, ID number, and CRC are not sent. This should only be used when the touch memory is the only device online since all devices respond to the phase-three Memory-Function command.

The Search-ROM command identifies all the touch memory devices online by a process of elimination. This command is used when the system needs to track all the touch memory connected to the system.

Once an individual touch memory has been contacted using either the Match- or Skip-ROM command, the third phase is entered by issuing a Memory-Function command. (The memory commands can be followed through the flowchart in Figure 5.) The entire or any small part of the touch memory may be read using the Read-Memory command.

A two-byte target address is set following the Read-Memory command to indicate which byte the dump is to begin with (see Figure 6). Only indirect writes to the touch memory are allowed. All writes are performed on a temporary scratchpad memory page and must be verified (read) before being copied to the appropriate permanent page.

The Write-Scratchpad command followed by the two target address bytes and data instruct the touch memory where to start writing the data within the scratchpad memory.

Verifying the scratchpad is done with the Read-Scratchpad command.

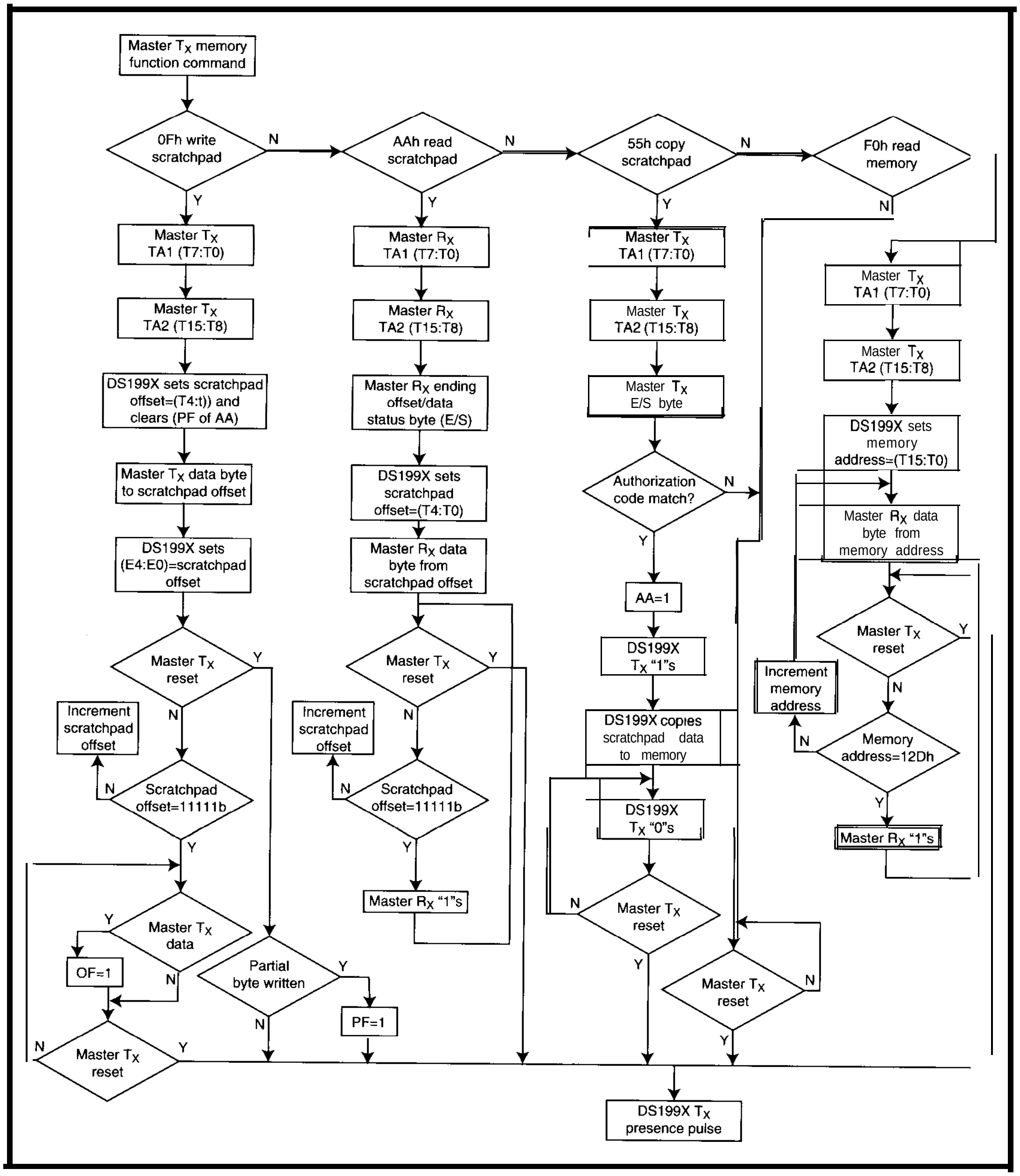


Figure 5—*The third phase of me one-mre communication protocol is the memory function command.*

The first two bytes read are the target address bytes, which are followed by the E/S register.

The E/S register indicates the last byte written to the scratchpad memory and the status of the write: OF (overflow-too many bytes written) or PF (partial flag-partial byte received).

The Copy-Scratchpad command can now be issued. Adding the two target address bytes and the E/S register initiate the copy. The E/S register also carries an additional flag-the AA bit. This bit is set when the copy command executes the copy function. The AA flag in the E/S register can be checked by using a Read-Scratchpad command. The AA FLAG is reset when a Write-Scratchpad command is issued.

	7	6	5	4	3	2	1	0
Target address (TA1)	T7	T6	T5	T4	T3	T2	T1	TO
Target address (TA2)	T15	T14	T13	T12	T11	T10	T9	T8
Ending address with data status (E/S) (read only)	AA	OF	PF	E4	E3	E2	EI	EO

Figure 6—*Three address registers are used to fell the DS1994 which page you're interested in and the status of the data transfer.*

EMPLOYMENT OPPORTUNITIES

Having touched on the available facilities and necessary protocol, let's look at some practical applications for this device. Dallas supplies a large quantity of code for the touch-memory product line. Code examples are given for a number of processors, which helps even if you're using a different micro (see Photo 2).

Temporary touch-memory holders are available for connection to a PC's serial port or parallel port [see Photo 3), which make setting the internal registers via your PC a breeze. In the field, a small laptop could be used to interrogate the device, making specialized equipment or software unnecessary.

In the most simple application, the DS1994 accumulates the time on. The device simply connects to ground and a logic high signal-you might wish to use V_{CC} or a control line to monitor its on time. This configuration is useful for items which may have a limited lifetime.

For instance, you may rent video equipment and need to charge your customers on an hourly per-use basis. Or, if you are responsible for an equipment crib, you could track hours of use for routine maintenance. Since all touch devices have unique ID numbers, inventories are also less frustrating. Copiers and other office machinery can be rented on a per page basis. The cycle counter enables the number of operations to be totaled for billing purposes.

When room for a bit of software is available in your application and you can spare a single I/O pin, you can make use of the DS1994's functions. In some cases, the DS1994 can even share a pin depending on the circuit design. Now, your circuit could log a time stamp along with its collected data.

If you needed to conserve power, the DS 1994 provides an interrupt to wake up your processor after a specified time period has gone by (the period could be minutes, hours, days, or whatever).

If your application has an active display, you can use the DS1994's NVRAM to hold status messages. These messages could be displayed whenever one of the alarm conditions arises from the DS1994's counters. These alarms could provide maintenance to be updated once maintenance is performed.

Although the 32-kHz clock accuracy is good for ±2 minutes a month (which is about 0.05% and similar to most other timebases), the accuracy can be improved by calibrating the counts over a known time period (the longer the better]. A difference between the proper count and the actual count can be stored in the NVRAM. If you are setting a real time or an interval time in the alarm registers, use the difference count to adjust the actual count being set. Likewise, when using the real-time counter to present the time and date, you might wish to update the real-time counter once a day with an adjustment based on the difference between the proper and actual time count.

THE WHOLE SHEBANG

Coming up with applications for this little tag is a whole lot easier than trying to figure out how Dallas fit all these features into this little can. The simplicity is elegant. The interface is inexpensive. The stainless-steel can is sealed for protection in harsh environments. Its totally self-contained lithium source provides energy for over one million memory manipulations and each can has a unique traceable ID number.

So, stop trying to put time in a bottle when it's already packed in a can.

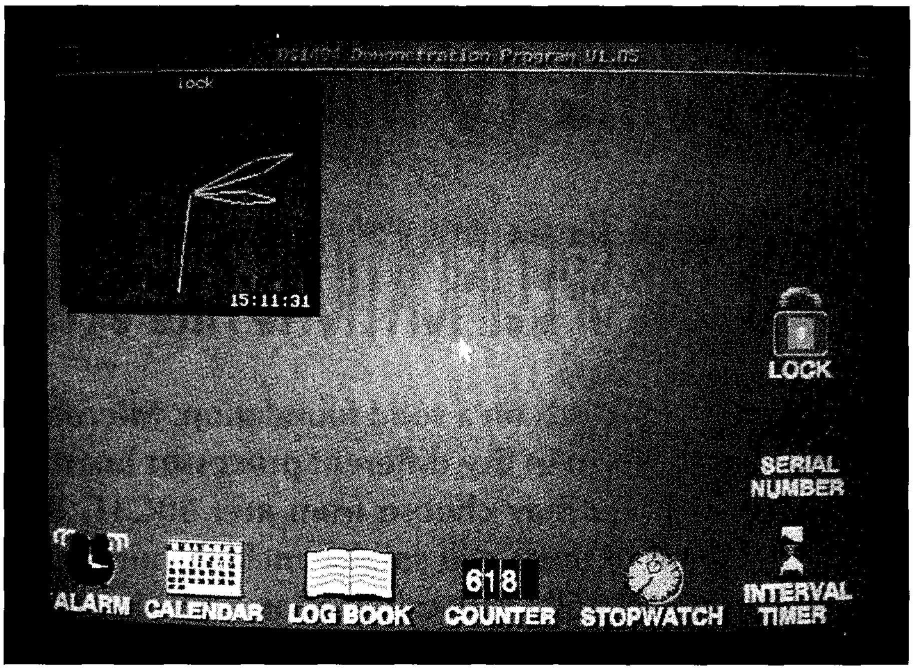


Photo 2-The Dallas Semiconductor PC demo software provides access to the DS1994 functions.

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

SOURCE

Dallas Semiconductor
4401 South Beltwood Parkway
Dallas, TX 75244-3292
(214) 450-0448
Fax: (214) 450-0470

I R S

416
417
418 Not Useful

Photo 3—Dallas Semiconductor can supply a wide variety of touch probes and sockets which easily interface to either a serial or parallel port on a PC.

EPAC Epoch

SILICON UPDATE

Tom Cantrell

Cursed by analog? Latch onto this new electronically programmable analog circuit. In Tom's opinion, this circuit marks the beginning of a new trend in computing analog relations.

It's surely testimony to the power of the entrepreneurial spirit that, even in the face of regulatory and financial roadblocks, most innovation comes from small companies. Mix one part blood, sweat, tears, seed money-and you've got the recipe for success!

It was upstart Intel that gave us the microprocessor and DRAM, not the then transistor-era powerhouses like Fairchild or TI. Apple and a bunch of S-loo-bus garage shops pioneered the PC years before big guns like IBM and HP had a clue. Tiny companies like MMI (Monolithic Memories, subsequently acquired by AMD}, Xilinx, and Altera evangelized programmable logic in the face of entrenched TTL naysayers.

Using 20-20 hindsight, it's easy to date the start of these epochs. Remember, at the time, it's hard to separate real breakthroughs from the hypes and hopes of those destined for extinction.

So, though I might end up with egg on my face, I predict that we're at the dawn of a new era. Let's call it the Digilog Revolution.

This month, we'll take a look at the Electronically Programmable Analog Circuit (EPAC) from IMP. Read on and see why I think the EPAC will go down in history as one of the opening shots of the revolution.

ANALOG GOES DIGITAL

I make no secret of the fact I'm an unabashed 1s-and-0s man. Sure, I know a volt from an ohm and can even wire up an op-amp if I must. Nevertheless, it's fair to say I'm what you might politely call "analog challenged."

Unfortunately, in accordance with a cosmic version of Murphy's Law, the real world is inarguably analog. Though the emergence of DSP and easy-to-use ADC chips push the digital frontier ever outwards, they only delay the inevitable. Ultimately, I end up on the edge of the Digilog Gap since whoever wrote the laws of nature forgot "thou shalt output O-5 V."

It might be easy to dismiss all this as the ravings of a bit-head bigot.

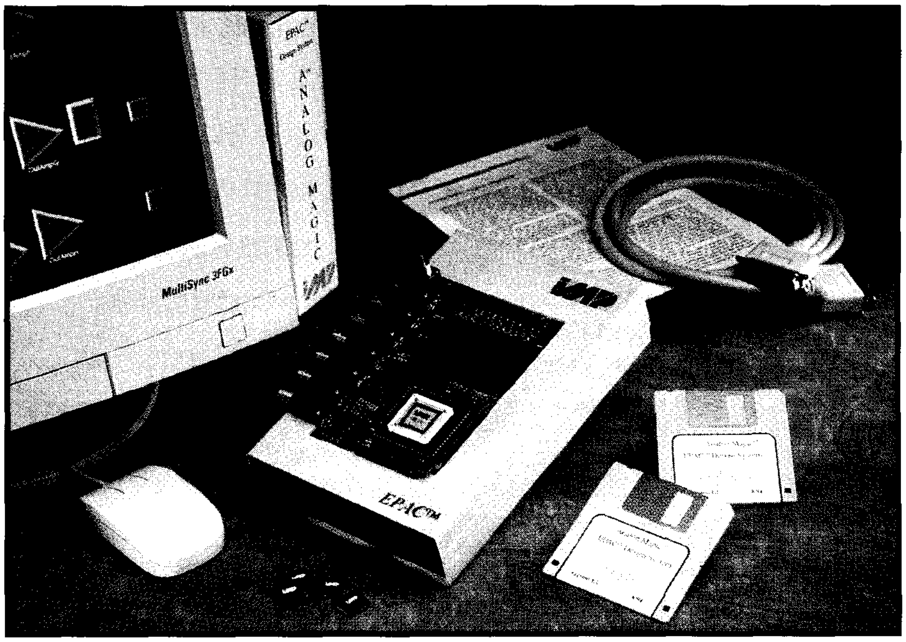


Photo 1—*The EPAC development system includes the Analog Magic software and a programmer/debug card that connects to a PC parallel port.*

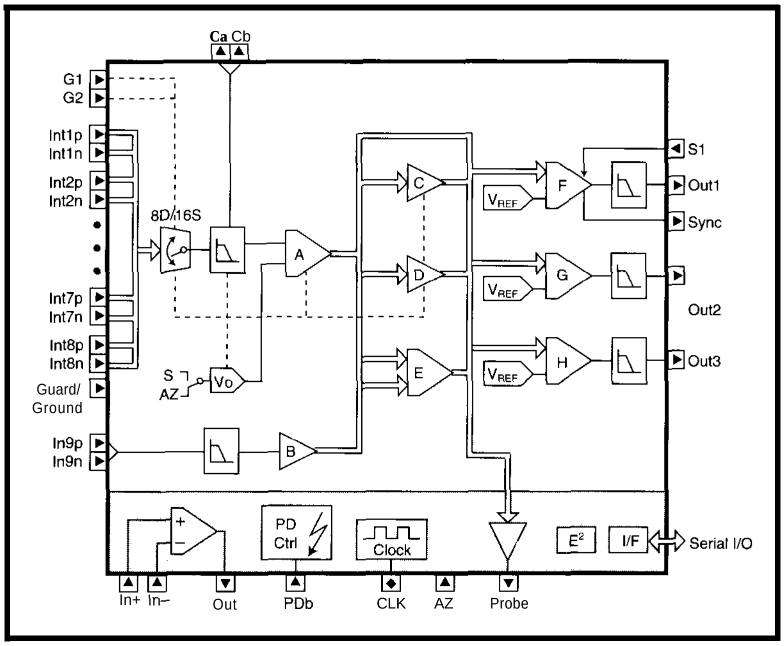


Figure 1—*The EPAC includes an entire signal-conditioning subsystem info a single chip*

Don't get me wrong. I have nothing but respect for analog wizards and the magic they perform with a hodgepodge of op-amps, resistors, and capacitors. Nevertheless, can even the wizards deny that casting spells is often an intricate and time-consuming ritual? It's true that even the most expert analog designers often end up humbled by balky circuits and seemingly endless redesigns.

Thus, I'm always on the lookout for handy gadgets to help bypass analog hassles, and the EPAC certainly fills that bill.

TWO CHIPS IN ONE

Few innovations can be characterized as immaculately conceived. Instead, most breakthrough products result from combining existing technologies in new ways. That's not to take anything away from the innovator, just to point out that identifying a need is just as important as cobbling a solution.

So, it is for the EPAC. It might best be described as an analog PLD. Inside, the EPAC contains a veritable data book's worth (18 in all) of analog macrocells including programmable-gain amps, comparators, multiplexers, DACs, filters, and others (see Figure 1).

The macrocells are connected via a so-called analog highway, the wide bus connecting the major modules. It looks a little confusing at first with everything appearing shorted together (e.g., modules C, D, and E inputs and outputs are connected). However, the diagram's intention is simply to show the universe of possible connections. A particular configuration of macrocell functions and connections is determined by the block labeled E2 (i.e., EEPROM) in the lower-right corner. A daisy-chainable, clocked-serial interface (I/F on the block diagram) serves as the communication channel for configuring and controlling one or more EPACs.

Does it all sound pretty familiar? If the cells weren't analog, such an organization would remind you of a PLD. The similarity is further hammered home by the company's PC-based Analog Magic design environment. As shown in Photo 1, the EPAC development kit includes the Windows-based software, a programmer/debug card (with handy scope connections), a cable [connects to the PC parallel port), and four chips. Best of all, the price is only $1169, which seems like quite a bargain.

Configuring an EPAC is easy—largely a point-and-click exercise. The screen presents an uncommitted block diagram showing all the internal macrocells. You're invited to wire them together as shown in Photo 2.

Defining the specific attributes of a cell is a simple matter of double clicking on it. You thereby open a cell-specific submenu (*see* Photo 3). Now tell me, isn't mousing around much easier than messing with a rat's nest?

Though pictured as a simple amp icon, the submenu hints that there's a lot of functionality hidden below the surface. In fact, the EPAC macrocells are as feature laden as can be and would probably sell well individually. Consider, for example, input amp A (Figure 2), which can handle either single-ended or differential inputs and features programmable gain of 0.5, 1, 2, 3, 4, 6, 8, and 10. It even has an autozero function that, within a few microseconds of triggering via the AZ pin or serial command, automatically nulls offset to within 20 µV.

AROUND THE CHIP IN 8 µs

It's easiest to understand the power and versatility of the chip by following an analog signal through the EPAC. Basically, the signal goes through three stages-input selection and conditioning, amplification, and output conditioning-from left to right across the block diagram.

Via the clocked serial port, a channel [i.e., one of 8 or 16 depending on whether differential or single-ended mode is configured) is selected for examination with the input multiplexer. Note that the inputs are logically subdivided into four groups. The group is selected either via serial I/O or two dedicated pins (G1 and G2). More on the usefulness of group switching in a moment.

Next, the chosen signal passes through a low-pass filter whose default (no external components) cutoff frequency is 15 kHz. Optionally, an external capacitor can be connected (Ca and Cb pins) to tune the cutoff frequency to your own liking. At maximum clock rate and considering the Nyquist (i.e., two-times sampling) limit, bandwidth is specified up to 125 kHz (i.e., an 8-µs sampling interval).

The filtered signal is shipped onto amp A, where it is combined with the output from the offset module. The latter is essentially a 10-bit DAC with programmable step size that can accommodate a wide range (from 20 µV to a whopping 2.54 V) of offset.

Meanwhile, two pins, a second input amp [amp B), and a filter combo are provided, even though they lack the group feature, external capacitor, and offset of the main inputs.

The conditioned inputs (A and B) go to the second stage, composed of amps C, D, and E. Besides gain programmability, these amps offer inverting or noninverting mode. Furthermore, amp E handles one or two (i.e., summed) inputs.

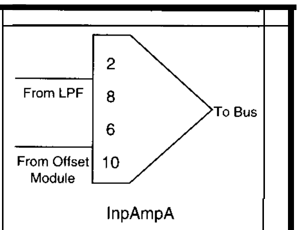
At this point, let's return to group switching. Note how the dotted lines from the G1 and G2 pins connect to the mux, filter, offset blocks, and amps C and D. The beauty of the scheme is that many of the settings and options, such as amp gain, internal versus external filter capacitor, and offset voltage, are switchable by group.

Moving on, the output stage is composed of fixed-gain (±2) amplifiers F, G, and H. The outputs drive nearly (within 0.05 V) rail to rail to maximize dynamic range. A 6-bit DAC (±2 V) provides the second (V_{REF}) input to the amps for further offset tweaking. Thanks to the DACs, the output modules can be configured as comparators (with optional 75-mV hysteresis] or simple voltage references.

Outputs feature a fixed 15-kHz (i.e., no external capacitor option) low-pass filter (these can be switched out if unneeded) and a turbo mode that boosts bandwidth and output slew rate at the expense of slightly higher power consumption.

Output amp F adds a track-and-hold feature with dedicated digital-control lines. S1 input triggers sampling. After acquisition and settling, the Sync pin signals a valid (i.e., held) output. Output droop is only 10 µV per second, giving even the pokiest micro

Input Amplifier Module A
Key features:
- Accepts single-ended and differential input signals
- Local auto-zero and l/f noise cancellation
- Gain choices available: 0.5, 1, 2, 3. 4, 6, 8, 10
- Utilizes group switching so that up to four different gain settings can be programmed, with the amplification level depending upon which group is selected.



PARAMETER	CONDITION	MIN	TYP	MAX	UNITS
Power supply current	Normal power		380	400	µA
	Low power		50	60	µA
Gain error	Gain = 1...10		0.3	1.0	%
Gain drift			tbd		
Input offset voltage	V_{CM} = 2.5 V, gain = 4		±1		mV
Input offset drift	V_{CM} - 2.5 V		10		µV/°C
Input impedance	Differential mode, normal power, gain = 4		20		MΩ
	Differential mode, low power		40		MΩ
	Common mode, normal power		2		MΩ
	Common mode, low power		4		MΩ
Nonlinearity	V_{inpp} = 3.5 V, 0–15 kHz			0.1	%
Distortion	0.5<V_{in}<4.5, 1 kHz, gain = 1	200	500	800	ppm
Input noise voltage	10 kHz		tbd		μV_{RMS}
Input noise voltage density	30–100 Hz	60	tbd		µV/√Hz
PSRR	@ 60 Hz				dB
CMRR	@0 Hz, gain = 4		55		dB

Figure 2-*EPAC cells* are powerful in *their own regard. For example, input amp A specs compare favorably with a stand-alone programmable-gain op-amp.*

plenty of time to take a reading.

POWER ME DOWN, SCOTTY

The EPAC, consuming a maximum of 20 mA (all modules active in turbo mode), is no power hog. Non-turbo mode cuts maximum power consumption further—to only 5 mA.

However, even a few milliamps quickly add up when it comes to battery-powered gadgets. So, the EPAC offers an ultralow-power sleep mode (70 µA max) under control of the PDb (Power Down bar) pin. Actually, you can choose whether power down is appropriate on a module-by-module basis, cleverly handling those applications in which a portion of the chip should be kept awake at all times.

The clock module controls overall chip timing and is bidirectional. In other words, an on-chip oscillator (nominally 500 kHz) can drive the pin, or an external clock can be input. In either mode, a programmable (1, 2, 4, or 8) divider helps match things up. For instance, using the divide-by-four option enables the EPAC to run off an external 2-MHz clock input or to generate a 125-kHz clock output from the internal 500-kHz oscillator.

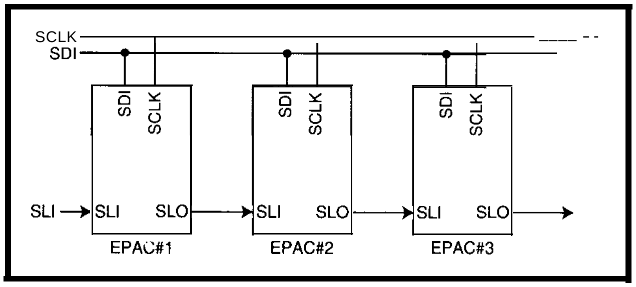


Figure 3—*The clocked serial interface offers daisy chaining of multiple EPACs on a three-wire interface.*

Another very handy feature is the dedicated Probe output. This is much like the other output modules, lacking only the fancy comparator and track-and-hold features. Under control of the serial interface, the probe pin can be connected to virtually any internal node along the analog highway.

Finally, as if that's not enough, IMP throws in an extra, uncommitted amp. Gee, I'm glad they couldn't find the kitchen sink!

With all the resources and routability EPAC offers, it certainly seems up to handling nearly any signal-conditioning challenge. For example, it's quite possible to cascade the intermediate amps (C, D, and E) to achieve nearly any gain factor, including all the way up to 20,000 (i.e., 10A x 10C x 10D x 10E x 2F).

It would be an interesting statistical exercise to figure out all possible EPAC configurations, but it's surely in the hundreds. Frankly, it's hard to imagine an EPAC getting stumped by any sensor.

A BIT OF PROGRAMMING

Field-programmable logic typically relies on either SRAM or EEPROM, each with its own advantages. For example, SRAM provides fast programming with unlimited endurance while EEPROM offers automatic power-up initialization without any extra

Group	Address (MSB...LSB)	Command Name	Function
xxxt nnnn			Write commands, with data following: (2nd byte = # of data bytes that follow)
	xxx1 0001	CR	Write subsequent data to Configuration Registers only
	xxx1 0010	CH	Write subsequent data to channel decoder only
	xxx1 0011	EE	Write subsequent data to EEPROM and Config. Registers
	xxx1 0100	SP	Write subsequent data to SoftProbe only
xxx0 nnnn			These are just commands, without any data following
	xxx0 0001	BP	Ignore subsequent data and set SLO low (bypass mode)
	xxx0 0010	RES	Reset and re-download from EEPROM
	xxx0 0011	PDG	Power-Down Global
	xxx0 0100	PDS	Power-Down Selective
	xxx0 0101	w u	Wake-Up (revert power-down states)
	xxx0 0110	EV	Enable VBGR Test-pin (only water level)
	xxx0 0111	SR	Set the readback registers
	xxx0 1000	AZ	Start Auto-Zero sequence
	xxx0 1001	CLA	Clear latches (reset only, no re-download)
any other			don't care

Figure 4-*The EPAC command set consists of single- and multibyte commands for configuration (e.g., SRAM or EEPROM programming) and control (e.g., channel and group selection).*

hardware or software. Thanks to the relatively small amount of memory required (160 bits), the EPAC is able to offer both, thereby achieving the best of both worlds.

All communication with the EPAC takes place over a simple shift-register serial bus that supports multi-EPAC daisy chains (see Figure 3). Data is clocked into the EPAC via the SDI (Serial Data In] line with SCLK (Serial Clock) at up to 1.5 MHz. Thus, loading a complete configuration to SRAM (referred to as the *configuration registers)* only takes a few hundred microseconds.

Programming the EEPROM is more than 1000 times slower, but that's still less than 0.5 s. However, the EEPROM's 10k write-cycle endurance limit dictates that the SRAM should be used for routine in-system tuning like adjusting gain to compensate for ambient temperature.

The SLI and SLO pins (Serial Load In and Out) manage the hand-off of data down the chain. When the first SLI input is driven low, data is shifted (but not latched) into the first EPAC, which decodes the command (see Figure 4) and determines how many bits to expect.

After the bits arrive, the first EPAC drives its SLO pin (and thus the next EPAC's SLI pin) low, loading subsequent data into the next device. The process continues until the last EPAC is loaded. Then, a high level placed on the first EPAC's SLI pin ripples through the chain, signaling each device to latch the previously shifted data and get to work.

Notice the absence of an SDO (Serial Data Out) or R/W pin (i.e., the serial bus is input to the EPAC only). This absence raises the question of how to verify the configuration, something which is a must during debug and production programming. Fortunately, the previously mentioned Probe pin, in addition to monitoring the analog highway, can also be used to dump configuration bits.

Piracy prevention is handled in the usual way with a security bit that, once programmed, disables configuration probing. The truly security conscious will be glad to hear IMP's claim that removing the chip's lid zaps the configuration info.

LUDDITES LAMENT

At $38 for singles ($20 at Ik) skeptics have the ammunition to claim the EPAC is just another expensive new-fangled gizmo, so it's better to stick with the tried, true, and cheap op-amps.

Of course, the same objection has been used-futilely-against every innovation since time immemorial. If doing things the old way is so great, how come we aren't still living in caves and scratching the dirt for seeds and roots?

EPAC isn't the final shot in the Digilog Revolution, but it certainly sets the stage for things to come. I wouldn't be at all surprised to see tomorrow's embedded systems

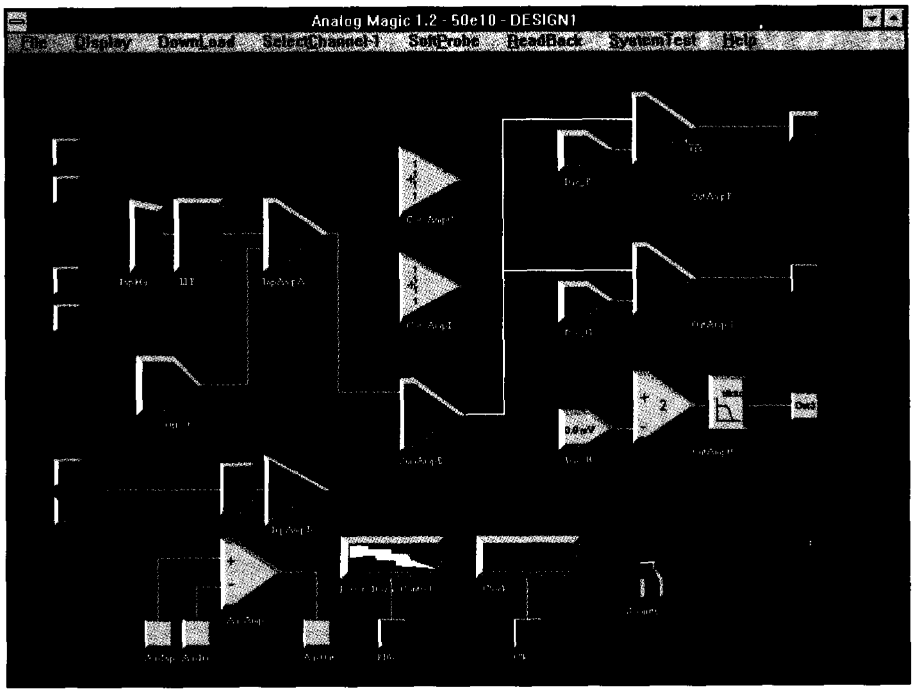

Analog Magic 1.2 - 50e10 - DESIGN1
File Display DownLoad SelectChannel-1 SoftProbe ReadBack SystemTest Help

Photo 2—*EPAC development starts with a* "breadboard" ready *for Wring." Modules* are highlighted *as they* are connected.

plastered with little "IMP Inside" stickers. ❑

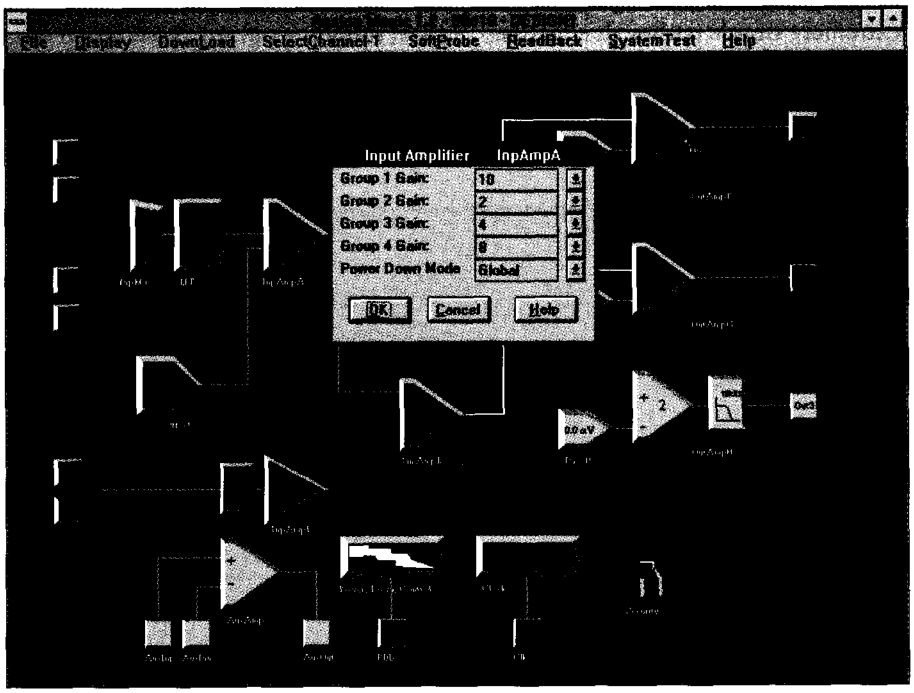
CONTACT

IMP, Inc.
2830 N. First St.
San Jose, CA 95134-2071
Attn: Ron Marfil, MS 115
(408) 434-1377
Fax: (408) 434-5904

Tom Cantrell has been an engineer in Silicon Valley for more than ten years working on chip, board, and systems design and marketing. He may be reached at (510) 657-0264 or by fax at (510) 657-5441.

I R S

419 Very Useful
420 Moderately Useful
421 Not Useful

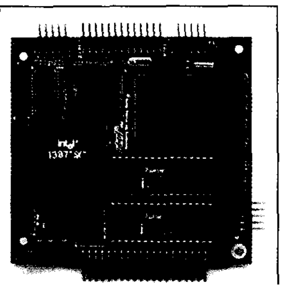

File Display DownLoad SelectChannel-1 SoftProbe ReadBack SystemTest Help

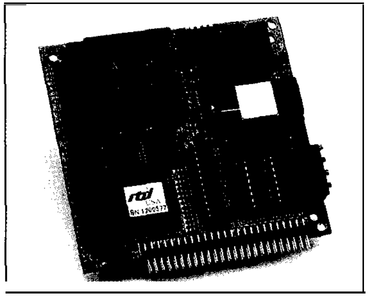
Input Amplifier InpAmpA

Group 1 Gain: 10
Group 2 Gain: 2
Group 3 Gain: 4
Group 4 Gain: 8
Power Down Mode: Global

OK Cancel Help

Photo 3—*Double* clicking a module brings *up* a submenu to tailor the options.

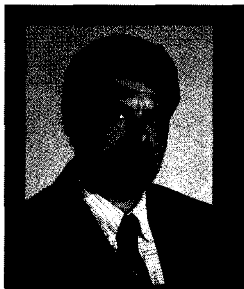




EMBEDDED TECHNIQUES

John Dybowski

Real Keyboard Emulation



John wraps up his AT keyboard discussion with a look at the command set, scan-code tables, and a real application. He emulates keyboard communication using touch memory and an 8751 microprocessor.

Oast month, I presented an overview of how the IBM keyboard works with a primary focus on the AT-style keyboard. My slant toward the AT was justified by a couple of factors. First, although there are a lot of PC/XT keyboards out there, there are relatively few computers of that vintage that you'd be likely to plug one of these emulators into. Second, since the PC/XT-style keyboard implements an extremely simple (although effective) first-generation protocol, there's not all that much that can be said about it.

Having been perhaps a bit hasty in getting to the actual keyboard-emulation code, I skipped over a lot of intermediate information. I touched on subjects such as how the PC and keyboard negotiate ownership of the line without adequately defining the actual command signaling used. Also, I demonstrated some working microcontroller-based keyboard-emulation code without first covering the correspondence between scan codes and the keyboard layout.

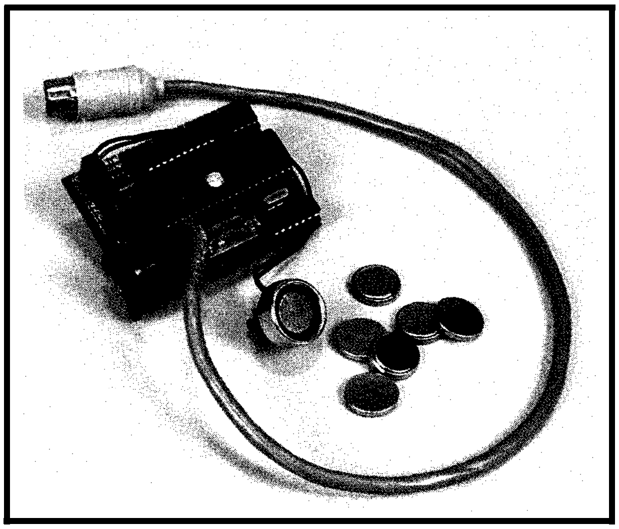


Photo I--The keyboard *wedge reads a Da//as Semiconductor touch* memory *device* and sends *the data to a PC/AT through the computer's keyboard port.*

This month, I'll fill in these gaps and wrap up by showing you how to combine several "standard" code modules to acquire data and "wedge" it into the PC keyboard interface.

By the way, the term "wedge" is not just my lingo. It is a trade term that refers to a class of products designed for data acquisition and computer entry via keyboard emulation. The acquired data can span anything from classic auto-ID applications for collecting data from bar codes and magnetic cards to applications involving analog sensors, precision measurement instruments, and standard RS-232 serial I/O devices.

More recently, "wedge" has been extended to include software products that capture data using the PC's serial port and make it look like it came from the keyboard without the need for any keyboard-emulation hardware. These products are especially attractive when you're faced with the prospect of jamming emulated-keyboard data into programs running on PCs that don't have detachable keyboards (i.e., laptop computers).

For obvious reasons, 1'11 concentrate on the hardware-firmware approach in this column. 1'11 begin by first rounding out my dissertation on AT-keyboard fundamentals that I started last month. Let's first look at what happens when you turn on the power switch.

POWER-ON RESET

The keyboard's circuitry generates a power-on reset (POR) when power is first applied. Documentation states that the power-on-reset interval may span the astounding range of 300 ms-9 s! How's that for loose tolerance?

Following this event of potentially glacial proportions, the keyboard controller executes a Basic Assurance Test (BAT), which is just IBM's way of saying power-on self-test.

The BAT includes the computation and verification of a checksum of all program memory and a test of the controller's internal RAM. During this time, the controller twiddles the keyboard's indicator LEDs on and off. The original IBM documentation reveals that the BAT runs anywhere from 600 ms to 900 ms. Frankly, considering the meager resources of the single-chip controller used in a typical keyboard, this must be one heck of a test.

Following BAT, the keyboard sends the completion code to the PC (assuming the line is not in inhibit status). This completion code is hex AA if the system checks okay. Interestingly, the documentation states that the error code is "hex FC (or any other code)" if a problem is encountered.

RESET (FF)-The PC sends this command to tell the keyboard to perform a firmware invoked "jump-to-O" reset. Controller execution then transfers to the power-on entry code and BAT.

RESEND (FE)-This command is sent if a transmission error is detected from the keyboard. It can only be sent following a keyboard transmission and before the system enables the interface. On receipt of this command, the keyboard retransmits the last byte sent.

NOP (FD-F7, Hex F2–EF)—If the keyboard receives any of these codes, it should respond with ACK and return to its prior activity.

SET DEFAULT (F6)—When this command is received, the keyboard resets all conditions to the power-on default state. Key scanning is not affected.

DEFAULT DISABLE (F5)This code resets all conditions to the power-on default states and causes the keyboard to stop scanning.

ENABLE (F4)—This command tells the keyboard to start scanning.

SET TYPEMATIC RATE (F3)—This command is followed by a parameter which the keyboard processes to change the typematic rate and delay. Note that the keyboard must first ACK the F3 command and then the actual parameter.

ECHO (EE)-When the keyboard receives this command, it responds with a hex EE.

INDICATOR CONTROL (ED)-The mechanics of this two-byte command are similar to F3 in that the ED command and then its parameter is ACKed. The parameter byte uses the three lower bits to control the keyboard's LED indicators. From least to to most significant bit, it indicates scroll lock, number lock, and caps lock. A similarly bizarre (from my perspective) set of commands can be sent by the keyboard.

RESEND (FE)-The keyboard sends this command to request retransmission of data on receipt of an invalid input or detection of a parity error.

ACK (FA)-This code is sent in response to any valid input other than the ECHO or RESEND.

OVERRUN (OO)-The overrun character is appended to the 17th (reserved) location in the FIFO buffer and is sent after all preceding codes have been output.

DIAGNOSTIC FAILURE (FD)-This code is sent if a diagnostic failure occurs during BAT. Also, the keyboard periodically tests the sense amplifiers and sends this code if a problem is detected.

BREAK CODE PREFIX (FO)-This code is prefixed to a scan code when a key is released.

DIAGNOSTIC COMPLETION (AA)-This indicates satisfactory completion of BAT.

ECHO RESPONSE (EE)-The keyboard responds to the PC's echo command with hex EE. An ACK is not transmitted in this case.

Table I--With the *PC/AT, commands at the fop may be sent from the PC to the keyboard while those shown at the bottom are senf from the keyboard to the PC.*

KEY SCANNING AND BUFFERING

The AT-keyboard controller continually scans the matrix keyboard, detecting keys pressed. It sends the appropriate scan codes to the PC in proper sequence regardless of the number of keys that are held down.

Inhibit status, when the computer jams the clock line low, results in the loss of the keystrokes during this condition. If the interface is not inhibited and not being serviced by the PC, the keyboard controller buffers up to 16 characters in an internal FIFO buffer.

Buffer overruns occur if more than 16 codes are placed in the FIFO buffer before the PC extracts the first one entered. In this case, code 17 is set to the overrun code-hex 00.

This seventeenth position is special and is reserved exclusively for the overrun code. Any more keys entered before the PC goes "live" are lost. When the PC starts accepting keyboard data, the codes in the buffer are sent in normal fashion with the overrun code denoting the place where key data was lost. Any new data is appended to the buffer's tail end.

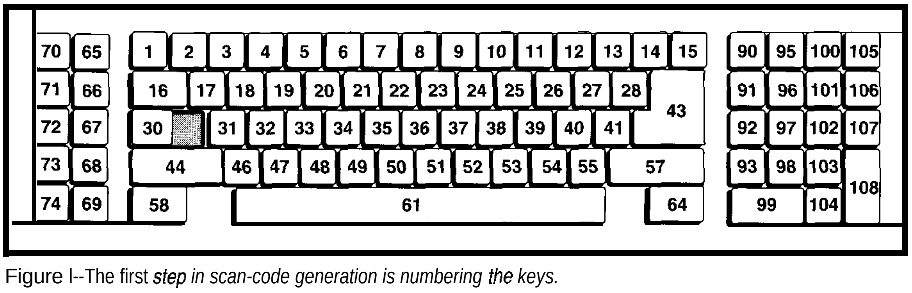


Figure I--The first *step in scan-code generation is numbering the keys.*

AT KEYBOARD COMMAND SET

A fairly comprehensive command set is defined for communications both to and from the keyboard. For most commands sourced by the PC, an acknowledgment (hex FA) is required. This acknowledge signal must arrive at the PC 20 ms after receiving the command.

In many cases, the command code has a different meaning depending on whether it is sent by the PC or the keyboard. Table 1 offers the complete command set.

AT SCAN CODES DEFINED

To recap, each key has a unique 8-bit scan code that is transmitted to the PC when a key is pressed and released. The release is accomplished by sending a hex FO break prefix followed by the scan code for that particular key. If the keyboard is operating in typematic mode, the typematic code is the same as the key's make code.

Table 2 illustrates the AT keyboard's key positions and the respective scan codes. The correspondence between key position and the keyboard's physical layout is pictorially depicted in Figure 1.

EMULATING KEYBOARDS FOR FUN AND PROFIT

Now that I've got the gruesome details out of the way, I'll demonstrate how keyboard emulation can be an advantage. For my demonstration application, I offer a device that accepts data from a Dallas Semicondcutor DS1990 silicon serial number and transmits it to the PC looking like keyboard data.

My circuit, shown in Figure 2, consists of an 87.5 1 microcontroller, its associated passive-support circuitry, a touch memory probe, a 74LS125 used as a buffer, and a line-switching relay.

l-DE	18—1D	36-33	55—4A	90-76
2-16	19-24	37—3B	56-51	91—6C
3—1E	20—2D	3842	57-59	92—6B
4-26	21—2C	39—4B	58-11	93-69
5-25	22-35	40—4C	60-19	94-77
6—2E	23—3C	41-52	61-29	96-75
7-36	2443	43—5A	64-58	97-73
8—3D	2544	44-12	65—D6	98-72
9—3E	264D	46—1A	66-DC	99-70
1046	27-54	47-22	67-OB	100—7E
1145	28—5B	48-21	68-OA	101—7D
12—4E	30-14	49—2A	69-09	102-74
13-55	31—1C	50-32	70-05	103—7A
14—5D	32—1B	51-31	71-04	104-71
15-66	33-23	52—3A	72—D3	105-84
16-OD	34—2B	5341	73-83	106—7C
17-15	35-34	5449	74-01	107—7B

Table 2—*Each key on the AT keyboard is assigned a number (as shown in Figure 1)* **and a** *corresponding scan code.*

The 8751 monitors the touch-memory interface looking for connection to a DS 1990 silicon serial number. Once a DS1990 is detected, it is instructed to dump its ID information. If this information is satisfactorily acquired, the 8751 seizes the line from the keyboard by energizing the relay, turns on the indicator LED, and transmits the DS1990's data to the PC as scan codes using an ASCII hex depiction. On completion of the sequence, the line is returned to the keyboard, the LED turned off, and the **8751** reenters the main DS1990 sample loop.

The touch-memory interface consists of nothing more than a touch probe and a 5.1 -kΩ pull-up resistor to +5 V. The keyboard interface is arranged so the 87.5 1 monitors PC and keyboard communication activity using a couple of port pins. These inputs are buffered using two gates of a 74LS125 buffer.

This simple implementation does not make use of this line-monitoring facility, but I provided the hardware for future development. Line monitoring is quite important since codes are transmitted between the PC and keyboard that drastically alter the way datastreams are interpreted and processed. What you don't know could most definitely hurt you.

To isolate the keyboard from the PC while the 875 1 is transmitting simulated-keyboard data, I use a DPDT relay. Remember that the AT-keyboard interface is defined as bidirectional and, as such, signaling must flow in both directions.

Needless to say, using a relay in such an application is not my preference, but it gets the job done. A more reasonable choice would involve an

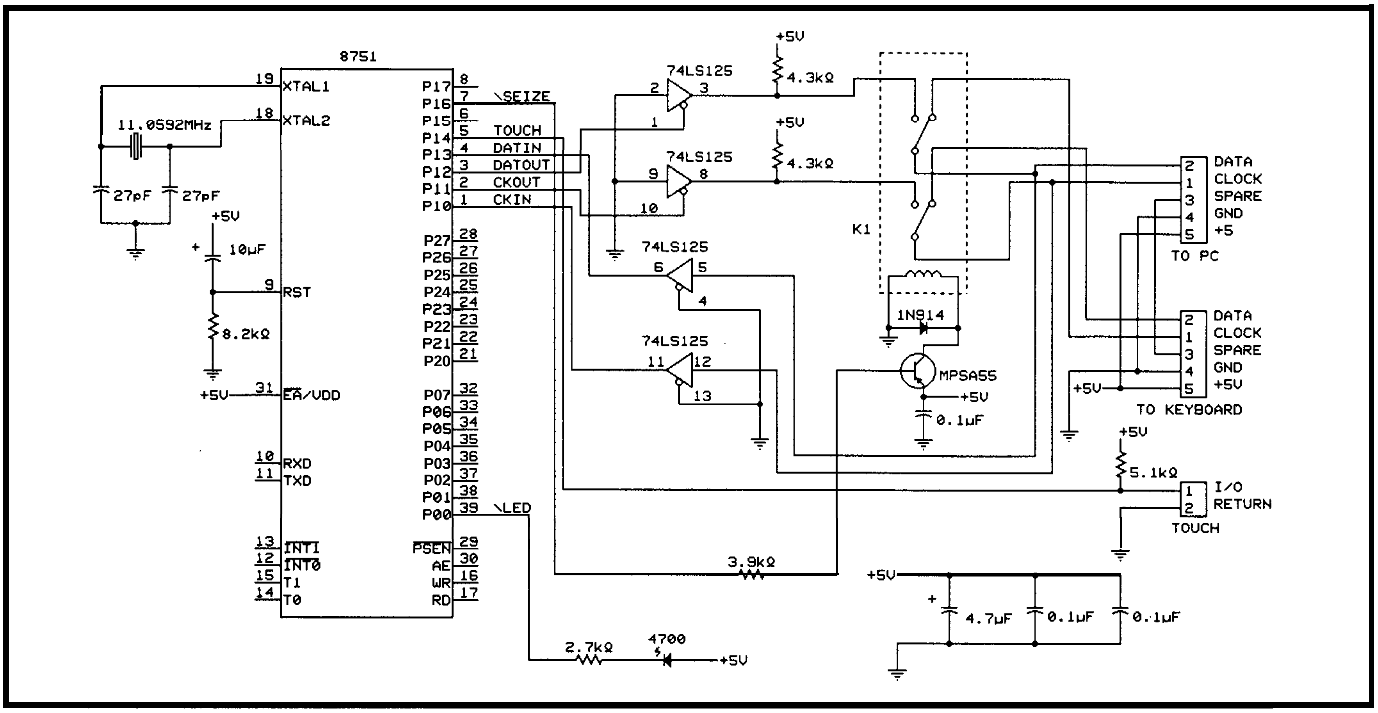


Figure *2-The keyboard wedge uses a relay to disconnect the real keyboard while it sends its data to the computer.*

Listing 1—*The* main *module coordinates* *the support functions.*

```
/* 8051 definitions  */
#include <8051reg.h>
#include <8051bit.h>

/* I/O bits */
#define led PO.0
#define seize P1.6

/* External references */
extern checkds1990 0;
extern getds1990 (register unsigned char *string)
extern atputstr (register unsigned char *string):
extern atputch (unsigned char a);
extern static char hextable[];

/* Global register (IRAM) data */
register unsigned char ds1990data[8];
register unsigned char atdata[24];

main (void)
{
  register unsigned char *s;
  register unsigned char *d;
  unsigned char c;
  unsigned char i;

  while (1){
    if (!(getds1990(ds1990data))){
      clrbit(led)
```

(continued)

analog switch or multiplexer. For additional information on using an analog switch in a PC-keyboard emulator circuit, see my Ondi remote-control article (*INK* 16).

Regardless of whether you use a mechanical or solid-state device to handle the PC-keyboard-wedge routing, there are several points worth mentioning. First, the switching system should connect the PC to the keyboard by default. Ideally, this should in turn be entirely dependent on the processor's power-on-reset default condition. That is, this condition should be established even if the 8751 does not come up running.

During normal operation, the PC and keyboard are directly connected. The only other circuits hanging onto the communication lines are the two 74LS125 input buffers that the 8751 uses for line monitoring. These are always connected so the 8751 can monitor the traffic between the PC and the keyboard and the PC and itself. When the 875 1 seizes control of the interface, the relay switches out the keyboard and switches in the local 74LS125 output buffers and their associated pull-up resistors. Now data can flow between the keyboard emulator and PC without disturbing, or being disturbed by, the keyboard.

Obviously, the simple application shown here could be done using a smaller processor than the 875 1. You could easily do the same thing using a low-end 805 1 derivative or any other small processor. I elected to use the 875 **1** primarily for ease of implementation. That I find it difficult to waste processor resources can be seen in Photo **1.** The additional circuitry you see on my prototype (not shown in the schematic) will not go unused for long.

JUST ADD CODE

With such a sparsity of hardware, it should be evident that the bulk of the functionality is rendered in firmware. The minimal wedge program consists of three main modules written in assembler and C.

Both languages have distinct advantages when used where they are

Listing I-continued

```
      clrbit (seize)
      s = ds1990data;
      d = atdata;

      for (i = 0; i < 8; i++){
        c = *s:
        *d++  = (hextable[((c >> 4) & 0x0f)])
        c = *s++;
        *d++ = (hextable[(c & 0x0f)]);

      *d++ = 0x0d;
      *d = 0;

      atputstr(atdata);
      setbit(led)
      setbit(seize)
      while (!(checkds1990()))


  return;


/* Binary to ASCII hex conversion table */
static char hextable[16] =

  '0','1','2','3','4','5','6','7',
  '8','9','A','B','C','D','E','F'
};
```

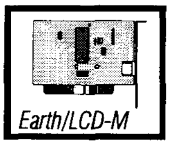
Listing 2—The keyboard emulation is *best done as a mix of C and assembler.*

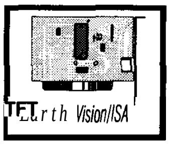
```
/* 8051 definitions  */
#include <8051reg.h>
#include <8051bit.h>

/* Key code constants */
#define kshift 0x012
#define kcontrol 0x014
#define kbreak 0x0f0

/* I/O bits */
#define clockin P1.0
#define clockout P1.1
#define dataout P1.2
#define datain P1.3

extern static char keytable1[];
extern static char keytable2[];

/* Translate null-terminated ASCII buf & send to PC as key codes */
atputstr(register unsigned char *string)
{
  unsigned char a;
  while (a = *string++)
    atputch(a);
  return:
}

/* Translate ASCII character and send to PC as key code */
atputch(unsigned char a)
```

(continued)

most effective. C relieves you of the tedium of having to keep track of bits and bytes when working at higher levels of abstraction. However, when you require super tight control at the bare metal level, then assembler offers the path of least resistance. That's not to say you can't coerce either language to operate out of its domain. But that's not the name of the game. Engineering is difficult enough. Selecting the wrong tool for the job significantly increases your hassle quotient.

In generating code for this column, I turn my attention to Dunfield's Micro-C 805 1 products. The developer's package provides everything you need to develop embedded programs for 805 1 family processors and includes a C compiler, assembler, optimizer, linker, librarian, a bunch of useful utilities, and a full library source of a variety of memory models, examples, and documentation. What amazes me most about this product is that the full package goes for about a third of what I paid for my first 805 1 cross-assembler alone!





Dunfield's package is a reminder to not let a price tag fool you. I've used a lot of the market's big-buck packages and Dunfield's compares favorably in efficiency and code size. As for the price, there is no comparison.

For this project, the C programs are compiled under the tiny memory model with all variables defined as "register" (internal RAM). Preprocessing is invoked to expand bit-manipulation capabilities, making these functions available directly from C. The C files are compiled down to assembler prior to being passed through the linker. The linker combines these (and other raw assembly) files with the appropriate start-up code and brings in the required run-time library functions. Running the output of this stage through an absolute assembler gives an Intel hex image.

The C language MAIN.C module, shown in Listing 1, is an endless loop, coordinating the operation of several support functions. On entry, the program continuously invokes the getds1990(*string) driverwhile waiting for DS1990 memory data to become available. The getds1990 function is passed the address of the destination buffer in internal RAM. On exit, the function returns the result code to the caller. A 0 indicates the availability of data.

If the program determines that DS1990 data is available, it falls through, turns on the indicator LED, seizes the keyboard interface, and sets up some variables in preparation for the simulated-keyboard transmission. Since the DS 1990 always returns eight bytes of data, the transmission length is fixed at this level.

The program now builds a null-terminated ASCII hex string and dispatches the converted data to the PC through the keyboard port using atputstr. The final stage involves turning off the LED, releasing the keyboard interface, and waiting for the DS1990 to disconnect. This last step is necessary since the system would continue to send while the DS1990 remained attached.

The assembly language DS1990.ASM support module (see the Circuit Cellar BBS for DS1990.ASM and INK

Listing *2-continued*

```
{
  unsigned char c;

  c = keytable1[a];
  if (c == 0x0){
    send(kcontrol);            /*  control  on  */
    send(keytable2[a]);        /* scan code     */
    send(kbreak);              /* scan code off */
    send(keytable2[a]);
    send(kbreak);              /*  control  off  */
    send(kcontrol);
  }
  else if (c == 0xff){
    send(kshift);              /* shift on      */
    send(keytable2[a]);        /* scan code     */
    send(kbreak);              /* scan code off */
    send(keytable2[a]);
    send(kbreak);              /* shift off     */
    send(kshift);
  }
  else {
    send(c);                   /* scan code     */
    send(kbreak);              /* scan code off */
    send(c):
  }
return:
}

/* Scan code transmission */
send(unsigned char c)
{
  delay (10);

  asm {
    JNB   clockin,$
    MOV   R0,#-5
    LCALL ?auto0
    MOV   A,[R0]

    MOV   C,PSW.0
    CPL   c
    MOV   R7,#9

    CLR   dataout
    NOP
    CLR   clockout
    MOV   R6,#20
    DJNZ  R6,$
    SETB  clockout
?LOOP
    RRC   A
    MOV   dataout,C
    NOP
    CLR   clockout
    MOV   R6,#20
    DJNZ  R6,$
    SETB  clockout
    DJNZ  R7,?LOOP

    SETB  dataout
    NOP
    CLR   clockout
    MOV   R6,#20
    DJNZ  R6,$
    SETB  clockout
  }
return:
}
```

(continued)

39 for a similar approach) has two entry points. The main routine is getds1990(*string), which performs all the steps involved in initializing the DS1990, command signaling, and data extraction using just one wire.

Since a one-wire protocol presents the very real potential for acquiring gibberish, a CRC is appended to the DS1990's datastream to ensure that valid data had been read. The getds1990 routine only returns a good result code if all criteria are met for data verification. A bad read is indistinguishable from a no-read as far as the caller is concerned.

The other routine in this module is chsckds1990( ), whichverifies that the DS1990 is not connected. The routine disconnects only if the initialization (presence detect) routine fails on 250 consecutive passes. This retrying should be enough to reject any contact chatter and indicates that the DS1990 is really not there.

The mixed C and assembler module AT. C, found in Listing 2, is the keyboard-emulation driver, which is based on the pure assembler keyboard driver I presented in last month's column. Providing public entry points for both character and string output, this module handles the initial table manipulations in C and drops into assembler to handle the low-level bit operations most efficiently.

This approach illustrates use of the respective languages strengths. There's no reason the entire program could not be written in C, and last month, I illustrated how it could be all done in assembler. The mix used here represents a reasonable compromise—a path of least resistance.

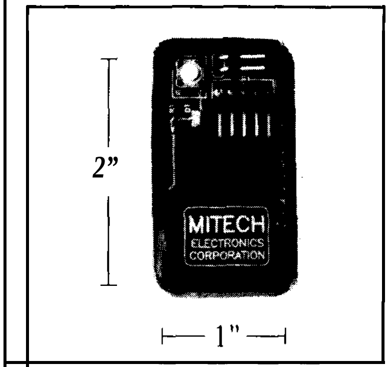
I did make one minor change to the fundamental logic I presented last month. As I pointed out, although the PC BIOS does not make use of most of the keys' break codes, you must realize that many (if not most) of the application programs on the market bypass the BIOS code entirely. Instead, they hook directly into the keyboard controller and make use of its output as they see fit.

It seems that certain programs interpret successive make codes as an

Listing *Z-continued*

```
/* Primary ASCII to scan code lookup table */
/* O=control character, Oxff=shifted, else scan code */
static char keytablel[128] =
{
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x66,0x0d,0x00,0x00,0x00,0x5a,0x00,0x00,
  oxoo,oxoo,oxoo,oxoo,oxoo,oxoo,oxoo,oxoo,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x29,0xff,0xff,0xff,0xff,0xff,0xff,0x52,
  0xff,0xff,0xff,0xff,0x41,0x4e,0x49,0x4a,
  0x45,0x16,0x1e,0x26,0x25,0x2e,0x36,0x3d,
  0x3e,0x46,0xff,0x4c,0xff,0x55,0xff,0xff,
  0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
  0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
  0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
  0xff,0xff,0xff,0x54,0x5d,0x5b,0xff,0xff,
  0x0e,0x1c,0x32,0x21,0x23,0x24,0x2b,0x34,
  0x33,0x43,0x3b,0x42,0x4b,0x3a,0x31,0x44,
  0x4d,0x15,0x2d,0x1b,0x2c,0x3c,0x2a,0x1d,
  0x22,0x35,0x1a,0xff,0xff,0xff,0xff,0x71
 };

/* Secondary ASCII to scan code lookup table */
/* All are legal scan codes */
static char keytable2[128] =
{
  0x1e,0x1c,0x32,0x21,0x23,0x24,0x2b,0x34,
  0x33,0x43,0x3b,0x42,0x4b,0x3a,0x31,0x44,
  0x4d,0x15,0x2d,0x1b,0x2c,0x3c,0x2a,0x1d,
  0x22,0x35,0x1a,0x54,0x5d,0x5b,0x36,0x4e,
```

(continued)



Listing *2-continued*

```
  0x29 0x16,0x52,0x26,0x25,0x2e,0x3d,0x52,
  0x46 0x45,0x3e,0x55,0x41,0x4e,0x49,0x4a,
  0x45 0x16,0x1e,0x26,0x25,0x2e,0x36,0x3d,
  0x3e 0x46,0x4c,0x4c,0x41,0x55,0x49,0x4a,
  0x1e 0x1c,0x32,0x21,0x23,0x24,0x2b,0x34,
  0x33 0x43,0x3b,0x42,0x4b,0x3a,0x31,0x44,
  0x4d 0x15,0x2d,0x1b,0x2c,0x3c,0x2a,0x1d,
  0x22 0x35,0x1a,0x54,0x5d,0x5b,0x36,0x4e,
  0x0e 0x1c,0x32,0x21,0x23,0x24,0x2b,0x34,
  0x33 0x43,0x3b,0x42,0x4b,0x3a,0x31,0x44,
  0x4d 0x15,0x2d,0x1b,0x2c,0x3c,0x2a,0x1d,
  0x22 0x35,0x1a,0x54,0x5d,0x5b,0x0e,0x00
};
```

indication that the keyboard has entered typematic mode. Apparently, they use this information to go into their own auto-repeat mode. Setting such a mode enables them to set the local typematic delay and the repetition rate independent of the actual keyboard controller's settings.

However, I found that without intervening break codes, certain PC programs append extra characters after receiving several identical make codes. Interspersing break codes was necessary to alleviate what, at first, seemed to be a bizarre firmware anomaly.

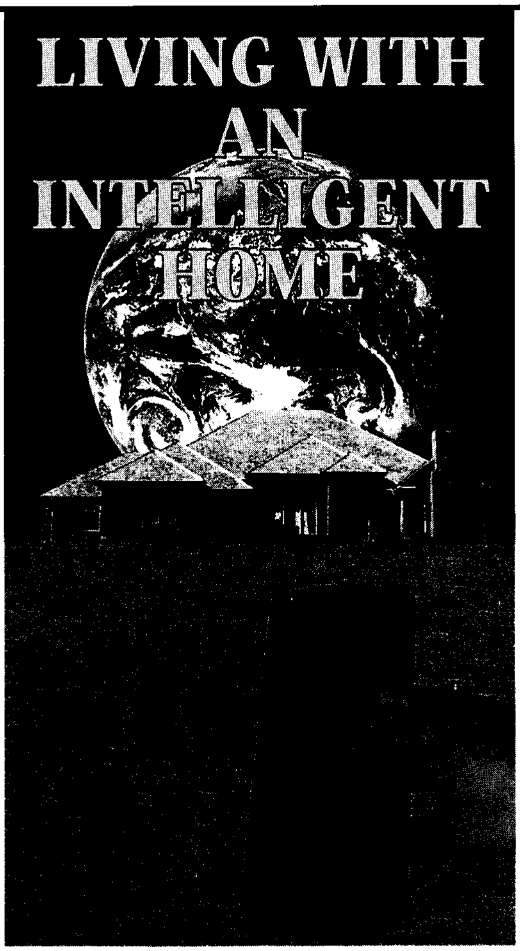
This example illustrates the pitfalls you may encounter when faced with the task of operating external devices with PC programs that do what you say rather than what you mean. Any serious keyboard-emulation application would be well served with a thorough survey of the various applications on the market. While such a survey is no small task, it represents a self-defense maneuver. There's nothing worse than finding out a year or two after you shipped your product that it really doesn't work.

Once again, I had to skip over much material due to lack of space. I've now got a breadboard and I may return to the subject once I've figured out all the ramifications associated with such an endeavor. In the mean time, don't forget those disclaimers. ❑

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.

IRS

422 Very Useful
423 Moderately Useful
424 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-I 988-Four incoming lines
Internet E-mail: **sysop@circellar.com**

We're going to stick to current and voltage issues in this month's message threads. First, we look at what if fakes to sense high currents in both AC and DC circuits. There are some interesting tricks available to accomplish the task,

In the other thread, we take a somewhat in-depth look at some of the cost and performance tradeoffs involved when frying to select between a cheaper part and a more stable and reliable design. In this case, it's choosing between an LM78xx regulator with trim resistors and an LM317.

Current Sensing

Msg#: *8172*
From: Ambrose Barry To: All Users

I am looking for some ideas concerning current measurement-medium to heavy, both DC and AC. I have a *little* info on Hall-effect devices, but no good app notes or ideas. I want to measure the current using an A/D voltage measuring board. Any help would be appreciated. Thanks.

Msg#:15908
From: **Pellervo** Kaskinen To: Ambrose Barry

There are a few basic ways of measuring current. Since you limit the applications to medium or heavy current, I won't go into the other end (electrometers).

A current shunt works on both AC and DC, but is more commonly used only on DC. This is due to historical issues, mainly the way available panel meters operated. A shunt produces a voltage proportional to the current. The nominal full-scale output used in the U.S. is 50 mV. In Europe, 60 mV is the common value.

At the low signal levels, an AC meter without an amplifier loses all the information into the rectifier drops. Nowadays, an amplifier is common and an easy solution, except for the additional battery or power supply required. In the old days, only copper oxide rectifiers were available to make the AC versions of moving coil (d'Arsonval) meters.

There are a couple of inherent limitations to the shunts. The first one is actually common with other high-current measuring systems as well-very limited frequency range. The physical size of the shunt comes with an inherent inductance that distorts the results with increased frequency. Typically, the specified accuracy of 0.25 % applies only to DC and possibly up to about 100 Hz AC. But note that it still is a good number; Hall-effect devices tend to be in the 1–3% range.

The other limitation for the shunts is the power losses. If we take a 50-mV shunt at 100 A, we have 5 W of power that the shunt has to dissipate. Take 1000 A and dissipate 50 W. Beyond that, it generally becomes unmanageable.

For pure AC, a current transformer is the obvious choice. It is a low-loss transformer, typically wound on a toroid core of µ metal. The primary can be a single turn, with the lead or bus bar just running through the center of the toroid. The secondary typically is an even number such as 100 turns. That gives a ratio of 100:1 (i.e., 100 A on primary forces a 1 A current on the secondary).

The normal rating for the secondaries is 5 A, so the above example produces 500 A primary full scale. Other secondary ratings are possible, with 1 A commonly used for long distances between the current transformer and the eventual panel meter. This is not so much to minimize losses in the first place as to allow a transformer with a lower "burden" rating to be used.

As I said, the primary current forces a proportional secondary current to run in the loop, regardless of the impedance there. But there is a limit to what the transformer can do accurately and without the core heating too much. It is like any transformer: a higher load capacity requires a bigger transformer. This capacity is called burden in the case of current transformers. Typical burdens are 20, 25, or 30 VA in 0.1 or 0.2% accuracy. A typical transformer also can support at least 5, maybe 10, or even 20 times that for short times in the relay actuation accuracy, around 2-3%.

The basic safety precaution with current transformers is to make absolutely sure that the secondary never gets open circuited while there is power on the primary. Imagine that the primary operates on 270 V, has one turn in the current transformer, and the secondary has 200 turns. If the secondary is left open, the core will saturate, but every time the primary current changes polarity, there is a voltage spike on the secondary. And guess what? The spike gets up to tens of kilovolts (maybe not quite to 200 x 270 = 54,000 but close enough to worry).

Instead of open circuits, a short circuit is perfectly legal on the secondary. You also could put in a MOV (Metal

Oxide Varistor) to clamp the voltage spikes. But you probably want to connect a diode bridge as the first element, then a load resistor. This arrangement eliminates the nonlinearity that the diodes represent, if used on the voltage signal. Now, the diodes are on the constant-current section and the transformer forces the current through them regardless of the 0.6-V drop they represent. The voltage you see at the terminals of your measuring resistor is a true representation of the current, to within 0.2% in favorable cases. You, of course, have to make sure the sensing resistor can handle the 5-A maximum current without losing accuracy.. .

Current transformers are cheap, very accurate, and do not require any separate power supply. Therefore, they are handy for all kinds of measurements in the tens to hundreds or even thousands of amperes. But they do have a shortcoming in what you seem to expect. They can not measure DC.

A variation of a current transformer based on a saturable core reactor can be made to operate on DC. But I doubt you are eager to build one, so I'll just skip it, especially since I'm ready to introduce what probably is ideal for your application.

A Hall-effect device has severe problems in reaching linearity and temperature stability if used directly. However, if it is used in zero magnetic flux condition, both problems go away. Of course, then we need more complicated structure and circuits. What it boils down to is a core with the gap for the Hall-effect cell, then a secondary winding of probably 5000 turns or so and a power amplifier that feeds the secondary winding with just the amount of current necessary to keep the magnetic flux balanced (i.e., equal to zero) all the time. The current in the secondary is a measure of the primary current and can easily be sampled on a low-power resistor.

Several years ago, when I was frustrated with some Hall-effect device limitations, I thought of this continuous balance principle. I even made preparations to make a prototype. But then we found there was already a commercial product based on the same ideas. That was LEM, a Swiss company. We have been using their products ever since, with only one minor gripe. They do not offer a "zero" remanence core material. Consequently, there can be a random offset of about 0.2-0.4 A on a 1,000-A rated device after a power down/power up cycle.

What you need with an LEM transducer is simply a ±15-V power supply of 200—350 mA capacity and a current-sensing resistor of 0.1–0.5% accuracy to get equivalent overall accuracies. The maximum output ("burden") is 10 V on the resistor terminals and the current requirement and resistance value are determined by the turns ratio in the sensor. They typically use 5,000 turns on the secondary side.

Nowadays, there are other vendors for similar devices, F.W. Bell being one and almost certainly Ohio Semitronics another one. But here is the contact info for LEM:

LEM U.S.A., Inc.
6643 West Mill Rd.
Milwaukee, WI 532 18
(414) 353-0711
(800) 553-6872
Fax: (414) 353-0733

I do not have the prices since they depend on the actual current range and physical options. Here are just some ballpark numbers for the versions we are using: 100 A, $50; 300 A, $100; 1000 A, $200. I hope this suits your budget.. ..

Msg#:16725
From: Ambrose Barry To: Pellervo Kaskinen

Thanks for the info, I'll digest it and get back to you. I had hoped for a simple Hall-effect solution. An old prof (I'm a semi-old prof) told me they would do the trick-but he didn't give me much help. I am looking for some app notes involving Hall-effect devices (used in the linear mode). Thanks again.

Voltage regs

Msg#: 1271
From: Kenneth Pergola To: All Users

What are the caveats, if any, of putting a resistor between a 3-terminal voltage regulator's ground pin and the common ground of the entire circuit. This will raise the output of the voltage regulator depending on the value of the resistance chosen.

Before I proceed, the idea is a cost-saving issue. An LM3 17 is the good choice for a variable voltage supply or a preset voltage supply, but it is more expensive than a 7805. I can only consider the option of taking an off-the-shelf 78xx and modifying its output as described above with only a resistor. No dropping diodes, zeners, and so on.

For instance, say the goal is to produce a 21-volt supply from a 7818 3-terminal voltage regulator. This device can be "tweaked" to get that output, but do any regulation parameters suffer-ripple rejection for example?

Somewhere I read that you could regulate high-voltage (e.g., 100 V) supplies with a 3-terminal voltage regulator as long as certain specs are maintained, such as keeping the

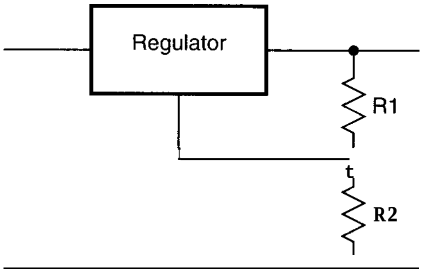
CONNECTIME

input/output voltage differential within the limits of the device. This has to be achieved with a resistance between the regulator's ground pin and the actual system ground of the circuit in question.

Msg#: 1336
From: James Meyer To: Kenneth Pergola

National Semiconductor's data books on voltage regulators show this. I guess if Bob Pease thinks it's OK, then it really *is* OK.

They recommend using a voltage divider instead of a single resistor, though. Like this:



The voltage regulator needs to use a little current from the raw voltage supply in order to work. That means there is a little current flowing in R2. The current won't be the same for different devices. The different currents mean that R2 will have to be selected and trimmed for each different regulator. If you add Rl to the circuit and make the current through Rl somewhat larger than the quiescent current for the regulator, then you can calculate the values of Rl and R2 and the output voltage won't change much when you swap out one regulator for another. Figure 5–10 mA for the regulator's quiescent current and 50-100 mA for the current through Rl. That means the current through R2 will be ±10% even though the regulator's quiescent current changes ± 100%.

I haven't heard of any problems with transient response from this arrangement. You probably *don't* want to add any caps across R2. The normal caps at the input and output to ground are always a good idea, as is the reverse-connected diode from input to output if the value of the output cap gets to be 100 µF or larger.

Msg#: 1377
From: Kenneth Pergola To: James Meyer

Gee, you are a gold mine! Thank you very much for taking the time to research this.

Texas Instruments seems to agree with what you found. I just found the sources that prompted me to ask the question to begin with.

Thanks for not steering me away from my goal. I'm developing a design which must cut costs. Sure I could go with the LM3 17, but in high quantities, the price difference between that beast (LM3 17) and a plain-Jane 78*xx* starts opening eyes!

Yes, I've been sampling various 78*xx* vendors; they are not as closely matched as I imagined they would be. The two-resistor design is a great way to circumvent these differences. I still may go with the one-resistor approach though; output tolerance isn't too critical. In any event, I'll do some careful evaluations before I commit to a PCB design.

So far things look pretty good. I'll have to check for anomalies caused by this approach with a scope. I've been warned that the chance of oscillation is greater.

One more thing, that diode protects for input shorts right? I know those devices are not input short protected. In other words, if the input voltage falls below the voltage present at the output the regulator may be damaged, right? I'm not sure I see when this condition could happen. Won't the input voltage always be greater than the output voltage? I've seen some designs with this protection diode, but more without it. I'm just wondering how necessary it is.

Msg#: 1387
From: Ben Mehlman To: Kenneth Pergola

Have you thought about putting a zener diode in place of R2 (the lower leg of the divider)? That would keep the ground reference of the regulator steady and it would allow the mismatched regulators to draw what they need.

Msg#: 1402
From: Kenneth Pergola To: Ben Mehlman

Thanks for suggesting the zener to replace R2 for the modified 78*xx* regulator. It is a good idea, but the one- or two-resistor approach should be all I need for this design.

I'm opting to go with the one-resistor approach because my output voltage doesn't have to be right on the money. The tolerance can be around ±0.75 volts. Thus, it seems that different manufacturers' 78*xx* regulators will meet that criteria.. .so far. If a batch of 78*xx* regulators exceeds that tolerance, a simple resistor change will suffice. National and TI seem to think it's OK to modify a 78*xx* this way.

Msg#: 1340
From: Lee **Stoller** To: Kenneth Pergola

The use of a resistor in the ground leg of the 78*xx* series can lead to problems with stability. It may work on the bench, but then give you all sorts of oscillation, poor regulation, and so forth at the worst possible times when your pet project is in operation. Why not stop worrying and

CONNECTIME

spend the extra buck? The 3 17 series is designed to do what you want. Use them.

Msg#: 1378
From: Kenneth Pergola To: Lee **Stoller**

I agree with your statement on using the LM317—it is an ideal solution. Its ADJ terminal, if bypassed with a cap, achieves a far better ripple rejection ratio than standard 78*xx* devices.

But perhaps you overlooked my goal in my post-it all boils down to economics. Of course *I* can afford to buy a LM317 for *my* projects, but when a company wants you to design to maximize the profit margin, the difference between the LM3 17 and a modified 78*xx* become quite apparent at large quantities. Reading my original post, I was vague and did not make it clear that production quantities may be involved.

Msg#: 1416
From: Lee Stoller To: Kenneth Pergola

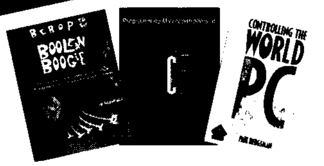
Well, OK. But I was really answering in economic terms, too, though I didn't make that clear. You can check your design on the bench, test production units, and still run into those dreaded failures out in the field! Mysterious failures that come and go with the phase of the moon and your latest bowling score. These problems are *very expensive* and will quickly erase any amount you save on parts. Parts are rarely the big expense in producing a product.

Msg#: 1421
From: Kenneth Pergola To: Lee Stoller

I concede, you are definitely right. I'm hearing the voice of experience from you-something I should heed!

I never really gave that too much, though. In certain cases, neglecting quality designing for economics' sake could lead to disastrous results: money out the window.

Thanks for driving that one home to me. I guess people would not mind paying a little more for a higher quality product. Moreover, I read that the LM3 17 has a superior ripple rejection ratio versus standard 78*xx* devices.



CONNECTIME

Msg#: 1422
From: James Meyer To: Kenneth Pergola

KP> I'm just wondering how necessary that protection diode may be.

Suppose you have a raw supply feeding more than one regulator or perhaps one regulator and some other load. Suppose the other load draws a lot more current than the load connected to the regulator. Suppose the regulator has a big cap on its output and the raw supply isn't filtered quite that heavily.

When the supply is switched off, the other load will pull the raw supply caps down pretty quickly, but the light load on the regulated side doesn't discharge the big filter caps on the output nearly as quickly.

That's what can cause a situation where a regulator can have more voltage on the output than it has on the input. It happens more often than you might think. Fortunately, most of the time the regulator survives. But if the filter caps on the output are large enough, and something else pulls the raw supply down fast enough...smoke.

When you do the board layout, leave room and pads for the extra resistor and diode. If you don't need 'em, you don't have to put 'em in. Holes is cheap.

Msg#: 1447
From: Kenneth Pergola To: James Meyer

I'm pretty sure I can leave the diodes out of my design, but putting the holes in the board just in case is a great idea.

My filter cap to the input of the reg will be 1000 µF and the filter cap on the output of the reg will be around 33 µF. The load only draws about 100 mA so ripple is not very significant. Full-wave rectifier.

According to my calculations:

Vripple = approx .8 V p-p = I/(f*C) = 0.1/(120Hz * 0.001)

I figure that the ripple rejection ratio of the regulator will let me get away with a 33-µF or less filter cap on the output.

The in and out will also be bypassed with 0.1-µF caps. Nothing else is connected to the raw supply.

Msg#: 1483
From: Pellervo Kaskinen To: Kenneth Pergola

I think you cannot afford to get out of tolerances as badly as the resistor-tuned 78*xx* series regulators would put you. Using your example of a 7818 for producing 21 V, there is at first the 0.8-V unit-to-unit variation. There is the 3–8-mA variation of the bias current with 1-mA max temperature effect to the bias current and 0.3-V line regulation without the resistor that probably would be increased with a resistor. If you are just making one system, then the selection process for the resistor might not matter too much, but if you indeed need to pinch the pennies out of the price, looks like you are considering some quantities. Then it becomes intolerable to have to trim each unit with an appropriate resistor.

Msg#: 1495
From: Kenneth Pergola To: Pellervo Kaskinen

Thanks for your reply. I've trimmed cost in other areas of the board design, so I should switch over to the LM317. As Lee Stoller pointed out, it can be a lot more costly in the long run if the modified 78*xx* becomes flaky.

In my design, I could probably use 5% resistors for the LM317—the output is not *too* critical. James Meyer looked up a National Semiconductor app note and they recommend using two resistors instead of one when modifying a 78*xx*.

My initial prototype uses a 78*xx* with one resistor; the board is working fine. But, I should probe around with a scope to ensure everything is clean. The PCB design will most likely incorporate an LM317, though.

I guess sometimes you can't put a price tag on peace of mind—the LM317 would give me just that. From your message, that is what I gather you are telling me right?

Msg#: 1627
From: Pellervo Kaskinen To: Kenneth Pergola

Peace of mind—yes! But also, in my experience, the cost of having to adjust every board even if you have a trimmer potentiometer is overwhelming. I tried to emphasize this aspect with the notion about unit-to-unit variation. Even if your application is not critical, the variation in the current through the single resistor is normally a killer. Admittedly, when you increase the natural voltage only a little, something like the 18–21-V (was it that?) boost, the two-to-one variation would only mean about 1.5 V on top of the 0.8 V that the chip itself presents. Maybe you could live with even that, but then maybe you could live with the 18 V in the first place? In any case, a 5% variation in the resistors is child's play in comparison with the 50% that the control-port-current variation represents.

Now, if your total current to the load is less than what you would need to put into a two-resistor network, then why bother? Let's see, if I remember correctly, you indicated something like 65 mA of load. To handle the up to 8 mA steering current with a 10% (or 20% worst case, including the two 5% resistor effects) maximum error, you would need to use at least 80 mA through the divider network, effectively increasing the "load" current to 145 mA. The 80 mA through an 18-V drop is about 1.5 W, meaning that your voltage divider ought to be built with a 2- or 3-W resistor. I just am glad that you have already consented to the 317-type regulator.

Msg#: 1675
From: Kenneth Pergola To: Pellervo Kaskinen

Thanks for knocking some sense into me Pellervo! It's nice to get help from the pros—even on such a "trivial" subject as 3-terminal voltage regulators. Some of these things you just can't learn in school—engineering is not all electronics. There is a lot of common sense and business sense in there as well. Adjusting many boards would be a nightmare—tedious as well as costly; what you say is very true. The LM317 is *the* way to go.

We invite you to 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.

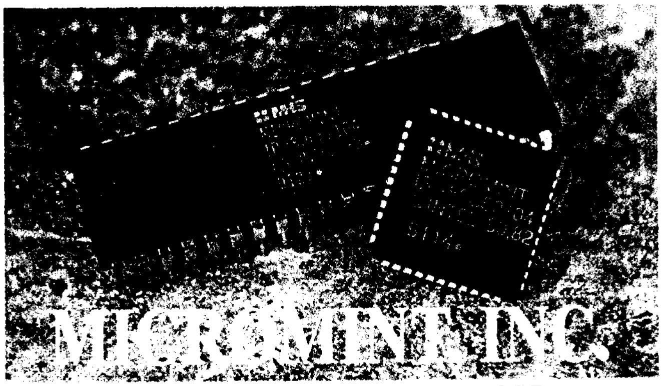
ARTICLE SOFTWARE

Software for the articles in this and past issues of *Circuit Cellar INK* 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: Circuit Cellar INK, 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.

I R S

425 Very Useful 426 Moderately Useful 427 Not Useful



STEVE'S OWN INK

Politically Correct Programming

for the life of me, I couldn't figure which one of us was to blame for picking Papa Gino's as a meeting site. Being the only guy in the place was bad enough. but dragging in all this junk and carrying on any kind of a substantive business meeting in a gaggle of screaming kids and harried mothers would surely be a miracle.

After three abortive attempts to find a table where there wasn't a castle made from grated cheese and hot peppers or various table games played using greasy pepperoni slices, we approached a small booth in the corner. Perhaps it's some vengeance thing when pizza becomes your whole life. but I detected a sadistic timbre in the waiters voice as he said, "This is all I have right now, sir."

I wriggled into the booth, fully realizing that I was probably going to become permanently bonded to it. As a necessary distraction, I whipped out a piece of paper and anxiously doodled on it while waiting for Jake. In preparation for our meeting, I drew a little schematic and proceeded to write a few lines of code. Finishing one side, I turned it over and wrote a couple instruction pneumonics.

Just then, a woman escorting her child to the rest room walked by my table. I had barely made eye contact with her when she scowled at me and exclaimed, "You Pervert!"

Instantly, I jerked upright, "Who? Me? For What?" Luckily, the din was high enough that no one else noticed. Fearing I had suddenly become front page news, I looked to see if I could understand the cause. All my clothes were buttoned. Wait a minute. On the top of my second doodle page, it said "SEX" in bold scribbles. Below it was "Get S".

After pronouncing her evaluation, she continued walking toward the rest room. I awkwardly tried to extricate myself from the booth as I yelled back, "Lady, wait a minute. You misunderstand. Really. That's Sign Extend! Not SEX like people. It's Sign Extend like in computers!"

Sweat was rapidly forming on my brow as I tried to contemplate how I was going to explain that "Get S" didn't mean "Get Suzie.' "it has to do with accumulators and registers, not children. Please, let me show you."

Perhaps because most perverts don't wear a jacket and tie to Papa Gino's or just that I looked so insanely plausible, she stopped and turned toward me. Instantly, I proffered a copy of *Circuit* Cellar *INK* and explained that I was merely writing a program. I showed her the schematic and the rest of the program and displayed how some of the abbreviations might look odd taken out of context. I thanked heavens that my program didn't contain any statements like Abort, Dump, Jump, Execute, Conditional SIN, Cleave, and On-Error Escape. It was hard enough rationalizing these few pneumonics.

Perhaps, like the technical wizardry she didn't quite understand in her new home-entertainment TV system, some computer things have to be taken on faith. She relaxed a bit. With a half smile and a conciliatory tone, she said, "You can see how anyone might jump to conclusions..."

My mind continued racing with impressions of how the Thought Police might want to restructure programming terminology so that innocent people, who stumble across an impure expression, might not be offended. Gone would be trigger terms like bus arbitration and bidirectional. instead, we'd be designing and programming with democratically negotiated reversible flow exchanges. I hesitate to even consider what they'd do with a master/slave relationship, never mind gender changers. Please save me from the alternatives.

Just as I had I finished expeditiously gathering my belongings and papers, Jake approached the table. With barely a breath between salutations, I emphatically said, "Let's go, Jake. Charlie's Steakhouse costs six times as much as this place. There are no kids and the Thought Police can't afford it!"

Jake looked at me in amazement, but actions like this were not new to him. He just chuckled and said, "Are you sure *I* can?"

Steve