CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

Using Intel's Flash RAM

The Theory of Control

Designing with Motorola's 68HC11

December '90/January '91 — Issue 18

$3.95

EDITOR'S INK

I've Seen the Future

Curtis Franklin, Jr.

I recently led a panel discussion at the Embedded Systems Programming Conference. I met interesting people, arranged for a few articles, and ran into lots of folks who were carrying crystal balls in their fanny packs. I thought you might like to know what they say you're going to be doing in the next few years...

OPERATING SYSTEMS

You're going to be using an operating system. I'm not talking about the operating system on your desktop computer, but the complete multitasking operating system that you'll be building into each and every control project. Depending on who you talk to, you'll be using an MS-DOS variant, a UNIX variant, or a specialized embedded OS for your development. All of these will make your life easier, your software more powerful, and your breath fresher. They may be right, but I'm not fully convinced.

First, an operating system assumes that you can accept a hefty chunk of overhead in your software. Anyone who's still forcing their code into 8K EPROMs will be in special trouble, but there will be more about that a little farther down the page. Most folks at the show weren't worried about memory constraint, and they weren't terribly worried about the cost the OS adds to the project. I've seen prices ranging from $99 for Coherent (a UNIX-like OS) to $3,000+ for some of the specialized embedded products. It's a different type of overhead, but substantial overhead increase all the same. Finally, an operating system adds another level of complexity to any debugging exercise. You can't just worry about how your software is interacting with the hardware, you have to consider how the passions in the hardware-OS-application software triangle are affecting one another. The time involved is still more overhead that must be absorbed by the project. All of this overhead is starting to add up, unless you're writing a truly large application.

LARGE APPLICATIONS

You know the 8K EPROM I mentioned in the last section? Well you can forget putting code into it. You're going to be writing embedded software that needs 500,000–1,000,000 lines of code. Stop for a moment and let that sink in.

Are you still with me? If you're going to spend the time, money, and effort to develop a million lines of embedded code, the overhead of an operating system gets lost in the background noise. Furthermore, I'm willing to say that, if you have a million-line program, you need an operating system to support your code. You also need a high-level language, heavy-duty libraries and support programs, and a debugger that will work with you in a most intimate fashion. Oh yes, you also need a processor that will let you get to that much code.

32-BIT PROCESSORS

There weren't many people talking about 8-bit processors at the show. Intel mentioned the 8051, saying that they have now shipped over 100,000,000 of them, but that came as a passing statement at their press conference announcing the latest members of the 80960 family. Motorola was ready and willing to talk about 68030 and 88000 applications, National was discussing the 32000, and everyone was announcing the arrival of the 32-bit future. Once in a while I heard talk about 16-bit chips like the 80186 and 8096, but there was a noticeable absence of discussion on anything having to do with 8-bit applications.

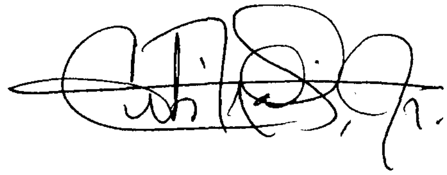
All of the predictions I've talked about are fine and probably true if you are involved in building avionics for the B-2 or automating a nuclear power plant. If, on the other hand, you're working on building automation, data logging, or day-today process control, 32-bit processors and million-line applications running under UNIX may be a bit of overkill.

CIRCUIT CELLAR INK

So what are we going to do? We're going to renew our commitment to 8-and 16-bit applications, with a special emphasis on stretching the limits of the possible with 8-bit processors. There are new 8-bit processors coming onto the market, and we will let you know about them. If you need more power, there is a lot of activity in 16-bit processors, and we'll keep you up-todate there. Operating systems will play a part in both 8- and 16-bit applications, as will high-level languages, so we'll look at how both can affect the project you work on. Finally, we'll remember that one of the reasons most of you read CIRCUIT CELLAR INK is that we take "the road less traveled" in our approach to applications. We're going to be going a little farther down those less-traveled paths in upcoming issues.

A NOTE

Scott Ladd's "Practical Algorithms" is taking a one-issue vacation. It will be back in CIRCUIT CELLAR INK #19.



FOUNDER/
EDITORIAL DIRECTOR
Steve Ciarcia

PUBLISHER
Daniel Rodrigues

EDITOR-in-CHIEF
Curtis Franklin, Jr.

MANAGING
EDITOR
Ken Davidson

PUBLISHING
CONSULTANT
John Hayes

ENGINEERING STAFF
Jeff Bachiochi
Edward Nisley

CONTRIBUTING
EDITORS
Thomas Cantrell
Christopher Ciarcia

NEW PRODUCTS
EDITOR
Harv Weiner

CONSULTING
EDITORS
Mark Dahmke
Larry Loeb

CIRCULATION
COORDINATOR
Rose Manse/la

CIRCULATION
CONSULTANT
Gregory Spitzfaden

ART & PRODUCTION
DIRECTOR
Tricia Dziedzinski

PRODUCTION
ARTIST/ILLUSTRATOR
Lisa Ferry

BUSINESS
MANAGER
Jeannette Walters

ADVERTISING
COORDINATOR
Dan Gorsky

STAFF RESEARCHERS

Northeast
Eric Albert
William Curlew
Richard Sawyer
Robert Stek
Midwest
Jon Elson
Tim McDonough
West Coast
Frank Kuechmann
Mark Voorhees

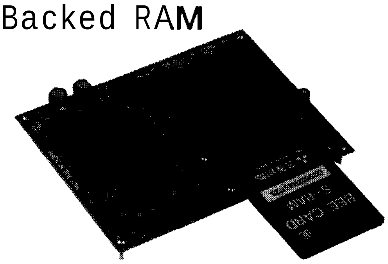
Cover Illustration
by Robert Tinney

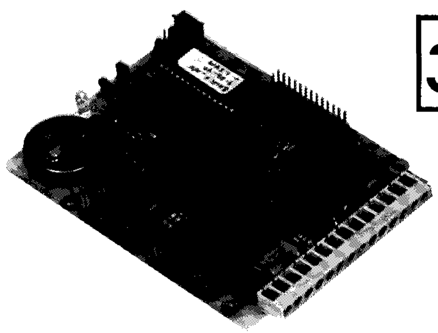
CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

In This Issue...

FEATURES





DEPARTMENTS

Circuit Cellar BBS-24 Hrs. 300/1200/2400 bps, 8 bits, no parity, 1 stop bit, (203) 871-1988.

The schematics provided in Circuit Cellar INK are drawn using Schema from Omation Inc. All programs and schematics in Circuit Cellar INK have been carefully reviewed to ensure that their performance is in accordance with the specifications described, and programsarepostedon theCircuit Cellar BBS for electronic transfer by subscribers.

CIRCUITCELLAR INK (ISSN 08968985) is published bimonthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (203) 875-2751. Second-class postage paid at Vernon, CT and additional offices. One-year (6 issues) subscription rate U.S.A. and possessions $14.95, Canada/ Mexico $17.95. all other countries $26.95 (surface), $38.95 (air). All subscription orders payable in U.S. funds only, via international postal money order or check drawn on U.S. bank. Direct subscription orders to Circuit CellarINK, Subscriptions, P.O. Box 3050-C. Southeastern, PA 19398 or call (215) 630-1914.

POSTMASTER: Please send address changes to Circuit Cellar INK, Circulation Dept.. P.O. Box 3050-C. Southeastern, PA 19398.

READER'S INK

1

Letters to the Editor

MILDEW REDUX

I wrote to you last year concerning the problem of mildew attacking diskettes which have been stored for six months or more in a tropical climate without air-conditioning. In your published reply (CIRCUIT CELLAR INK #11), you could not offer a remedy but asked that we write back if we found a solution. Well, it does seem that we have solved the problem since nearly a year has passed and we have not had a recurrence. The solution found was neither the use of desiccators nor of fungicides but the use of a cupboard kept a few degrees above room temperature. This remedy was suggested by a friend who said it was used by photographers to store stocks of film and keep them free of mildew. It seems to work for diskettes as well, with no bad sideeffects. The cupboard is warmed by a bulb of about 10 W (I use either a 60-W 240-V bulb at 120 V or two 25-W bulbs in series as I could not get a 10-W bulb).

I suspect that others living in humid climates may have this problem but without recognizing it. If you examine the surface of your disks (especially back-up copies) in reflected light, mildew appears as light blotches-it is difficult to see otherwise. Frequent use keeps it at bay for some time, but may eventually lead to disk errors. Continuous air-conditioning no doubt also prevents mildew.

Andrew Mancey
Guyana, South America

BACK TO THE COMEFROM

The article by J. Conrad Hubert entitled "Implementing a ComeFrom Statement" (CIRCUIT CELLAR INK #15) brought back memories of a project I led in 1980 to develop a microcomputerized controller for the paging system at Lambert St. Louis International Airport. We wanted to trigger the interrupts of a 6502 microprocessor (part of an AIM-65 board) from a sanity timer to restart the system in case the processor got off track and wasn't executing properly. We had the same problem: Once we got the microprocessor's attention, how would we get it to go back to restart instead of back to where it left off when the interrupt occurred, without causing stack problems? The only logical thing to do was to modify the return address on the stack.

This method, while very practical, would probably be frowned upon by those adhering to strict structural techniques, especially those who develop software for military applications (per standards like DOD-STD-2167). However, if it is viewed as a form of exception handling, then it would probably be considered "respectable."

I'm not sure, however, that this method helps "discover where your code has been," as your subtitle to the article suggests, but rather, it "changes where your code is going to."

Kenneth J. Ciszewski
Overland, MO

LEGO CONTROL

I'd like to comment on the letter "Keep those Legos Moving" in CIRCUIT CELLAR INK #16. The little motor driver circuit that was described is essentially available as Sprague UDN-2952B or UDN-2952W full-wave bridge motor drivers. They are available from Circuit Specialists, Mesa, Arizona. They are capable of 3.5 A of output current and feature thermal protection. They require a 5-V and a motor power supply. Typically, the control signals are a DIRECTION and a NOT ENABLE line. The use of a NOT ENABLE line prevents the motors from running when the control lines are disconnected. Remember that TTL inputs float high. It is also possible to drive the NOT ENABLE line with a PWM signal to allow speed / torque independence.

I have successfully used these to control some small DC motors. DC motors have an overrun problem when stopped. In order to minimize this, I added a relay and some logic which would short the motor winding when stopped. This provides very quick dynamic breaking. The motor is acting as a generator into a short circuit and therefore stops rather rapidly.

Ron Dozier
Wilmington, DE

ON ANOTHER LEVEL

I have been "into computing" for quite a few years now: My first computer job was in 1957, preparing data for the IBM computer operated by Hughes Tool in Houston. I worked on mainframes (IBM 7094s, 360s, 370s, and HW2000s) for several years, while wanting a computer of my own. I finally managed to "graduate" to micros in 1981 when my boss got me an Apple II+ (at last, a computer of "my own").

I have been mostly a software person for many years (application and systems programmer and systems analyst, currently teaching programming at a community college), but I have occasionally messed around with electronics (yes, I know my way around a soldering iron)—my call sign is KA7PHM.

I subscribed to CIRCUIT CELLAR INK because I hoped to find a place that would give some tutorials on computer circuit design-such as how to go about putting together a computer "from scratch"-you know, what chips are needed, what ones go together, what the significance of microprocessor timing diagrams is, how to interface to memory (and get the RAS and CAS signals timed right), and so forth. In short, I need help to get me going (with some assurance) in hardware.

What I have in mind is to learn enough to try to design a microcomputer I have wanted to build for several years now, so I can try my hand at writing the system software for it, and.. .(the stuff dreams are made of, you know!)

It would be nice to get some help. Of course, for you to run "tutorials" might alienate some of your other subscribers for the same reason I have gotten tired of some of the other computer magazines that seemed to run yet another "Getting started in BASIC" (or assembler, or C, or.. .) article every 18-24 months.

I wonder if the information I am looking for can be found in the books of Ciarcia's Circuit Cellar articles, or whether I should get some other (more theoretical?) books and try a few projects (like Don Lancaster's "Cookbooks" perhaps)? Any adviceand direction you can provide would be appreciated.

Richard P. Winslow
Riverton, NY

When CIRCUIT CELLAR INK was started, we discussed the idea of presenting articles for computer novices. We decided that there was no way to do that without "watering down" the content for our core of readers: experienced engineers and programmers. We do print tutorials, but they're&signed to "teach old dogs new tricks," and not to give folks an elementary education.

There are many books that contain the sort of information you're looking@. Ciarcia's Circuit Cellar, Volumes 1-7; Don Lancaster's Cookbooks;and any number of titles from Tab Books canget youstarted down the road tomicrocomputerand controller design. Good Luck!

PHONE WIRE IAN

I was reading through CIRCUIT CELLAR INK #17 when I ran across "Low-Cost LAN using X-10 Technology" in the ConnecTime section.

There is an excellent base for a low-cost **home** LAN using the existing telephone wire. Each jack has an IW (inside wire) feeding it. The IW is either a 2-or 3-pair cable. The phones only require a single pair (red / green), leaving the yellow / black pair available for a low cost 2-wire LAN. Most homes do not have the second telephone line installed, so all that is required is that the yellow / black pair be disconnected from the jack used by a telephone. This is required since many phones use the yellow / black pair for switch hook signaling (A and Al leads). I have used this second pair for quite some time with no major problems. The average home **hasbeen prewired** with five jacks, so the odds are pretty good that there is one near the PC or desired location at this time.

I have a suggestion to make so that the LAN can be used by more than one device: Transmit the data on a single wire (yellow) using the last wire (black) as a circuit busy wire. This way, when you have multiple devices connected they will not access the data line at the same time.

Don Houdek
Spring Grove, IL

ONE FOR MODULA-2

This letter is about the choice of language for Scott Ladd's column on Practical Algorithms, and responds to Mr. Don Lasley's letter to the editor in CIRCUIT CELLAR INK #17. Please enter my vote for Modula-2 as the choice for algorithm exposition.

C is not more powerful than Modula-2-anything, including bit-twiddling, that can be done in C can be done in Modula-2. Of course, you have to know Modula-2 as well as you know C in order to be able to do the same things equally easily. Mr. Lasley can't complain that the language he hasn't bothered to learn, is clumsier or harder to use than the language he's used extensively.

As a principal engineer performing quality control for a major consulting engineering company, I review work by a large number of engineers and analysts. Based on this exposure, my conclusion is that there is no doubt that, while C is great for quick-and-dirty hacks, nobody, including the programmer, should expect to go back later and understand what was done. Programs written in Pascal, and especially Modula-2, on the other hand, tend to be clearer and better organized, making them easier and faster to review and verify. They are also more likely to produce reusable code.

Stephen R. Troy
Severna Park, MD

HOME AUTOMATION CATALOG ON FLOPPY DISK

Home Control Concepts has released its newest catalog on diskette. The 1990 **Home Automation Catalog on Floppy Disk** includes home automation and security equipment, specifications, pricing, explanations, and uses. The catalog also includes information on home automation, the CEBus (Consumer Electronics Bus) home automation standard, SMART HOUSE, and industry definitions. The catalog is menu driven and has illustrations of home automation and security equipment. An IBM PC or compatible computer with graphics is required.

The unique diskette format allows for dynamic changes, so that each catalog distributed may have the most current information, prices, and new products. Additionally, far more information can be distributed on diskette that can cost-effectively be distributed via paper mail.

The catalog is available from Home Control Concepts for **$1.00** (refundable with any product order) or free from several computer bulletin boards including GEnie.

Home Control Concepts
9052 **Westvale** Rd.
San Diego, CA 92129
(619) 484-0933

Reader Service #500

FRACTAL GRAPHICS SOFTWARE

A drawing program that uses fractal geometry to create complex images is available from Cedar Software. The program, **"Fractal Grafics,"** can be used for graphic design, scientific visualization, and educational illustration as well as desktop publishing and presentations. Fractal Grafics takes a simple template drawn by the user and automatically continues the pattern. For example, the user draws the trunk and first few branches of a tree, and the program will draw the rest. A mouse or keyboard can be used to rearrange parts of any shape without losing texture or detail. The changes can then be reflected through all levels. The program features an on-line interactive tutorial, point-&click menus, and full color control.

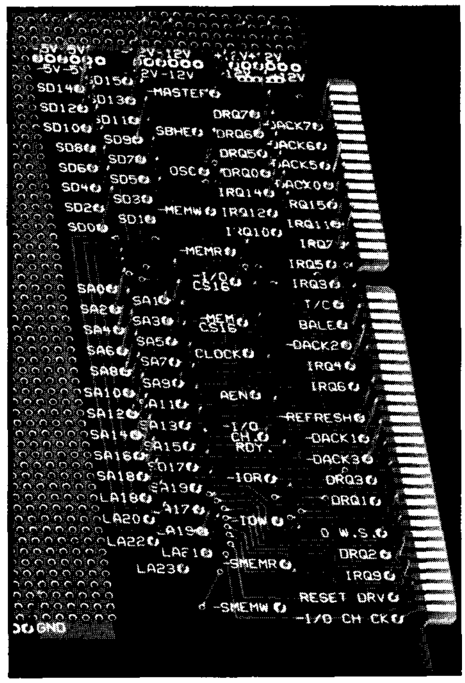
A 120-page guidebook and over 150 hands-on examples explain and illustrate fractal art, science, philosophy, mathematics, history, and literature. Example templates, which can be modified as required, are included. Chaos theory and the formulas used to create fractals are explained, and a program to draw the famous Mandelbrot and Julia sets is included.

Fractal Grafics works on IBM PC compatibles with 384K RAM and CGA, EGA, or VGA displays. A mouse and math coprocessor are optional. PCX file compatibility allows the exchange of images with all major graphics programs. The program sells for $79.00 and includes one free program update.

Cedar Software
R1 Box 5140 • **Morrisville, VT** 05661
(802) 888-5275

Reader Service #501

WIRE-WRAP PROTOTYPE CARD FOR PC/AT



A manually wrappable prototype card for IBM PC/AT and compatible computers has been announced by CANA Group. The Protosystem **AT** has been designed to simplify prototype design and save debugging time. The board features wire-wrap pins soldered in place on every signal line, tantalum bypass capacitors on every power line, and holds more that 100 16-pin IC sockets.

The board is designed for ease of use and minimization of assembly errors. The signal pins are never closer together than 0.200" (double the normal distance). The signal lines are grouped into address, data, and control buses, and each signal pin has a large, functional name label. The labels are on both sides of the board for ease of troubleshooting.

The board deals with a number of bugs that plague high-speed circuits by eliminating the conditions that cause them. The hand wiring can be used to reduce the chance of cross talk by using several levels of loose wiring rather than a neat bundle. The power and ground planes are low-inductance paths to the power pins around the edge of the board that reduce losses. The large number of locations for power pins eliminate the need for daisychaining and long power leads, thereby reducing voltage drop and noise propagation.

A manual, entitled 'The Principles of Wire-Wrapping" is included with the board. This brochure describes techniques in color coding of wires, socket ID labels, and test points to make a circuit much easier to test and debug.

The Protosystem AT is available from stock and costs **$149.95.**

CANA Group
100 Walnut St., Suite 402
Peoria, IL 6 **1602**
(800) 747-2262

Reader Service #502

PATTERN RECOGNITION BOARD

A pattern recognition board, featuring 24 TTL/DTL digital I/O lines, is available from Keithley MetraByte. The PIO-INT is an IBM PC/XT/AT compatible board that has been designed for use in data acquisition and control applications, and consists of three byte-wide ports provided directly from a standard 8255 PPI.

On-board circuitry monitors two of the three ports and is capable of generating interrupts on either the change of any bit in the port(s) or when a specific bit combination or pattern appears. With a bit interrupt, a change of any unmasked bit from either 0 to 1 or 1 to 0 will generate an interrupt. Changes can be read from the Status Register, and only those bits that are activated by the interrupt Mask Registers will generate an interrupt. With a pattern interrupt, an interrupt is generated on a given pattern of bits in any port. Only unmasked bits can participate in a pattern match interrupt, and they are compared with a stored pattern of bits in the PIO-INT's Pattern Match Registers.

The third port, while not monitored and without interrupt generation capability, can be used as an auxiliary I/O port, and can be divided into two nybble-wide ports. Each port can be individually configured as an input or output port.

To prevent spurious interrupts, generation of an interrupt is delayed by a two-stage digital filter. The filter is clocked by a programmable discrete decade frequency ranging from 1 Hz to 10 MHz. The effect of the filter is to allow actuation delays from 100-200 ns to 1-2 seconds before interrupt generation.

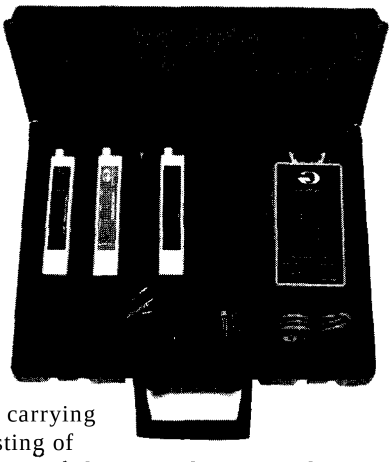
Filter delays are selectable independently for the two ports.

The PIO-INT is provided with a sample assembly driver callable from BASIC, together with its source code listing on a utility disk. The PIO-INT sells for $399.

Keithley MetraByte
440 Myles Standish Blvd.
Taunton, MA 02780
(508) 880-3000
Fax: (508) 880-0179

Reader Service #503

DIGITAL TROUBLESHOOTING KITS



Three digital logic test kits which are TTL and CMOS compatible have been announced by Global Specialties. The LTC-6, LTC-7, and LTCJ each feature a logic probe, LM-2A 16-channel logic monitor, DP-1 logic pulser, and SQ-1 tone-ohmmeter packaged in a rugged plastic carrying case. An accessory kit, consisting of interchangeable probe tips, ground clips, tip adapters, and quick hook cables is also supplied.

The tone-ohmmeter locates bad ICs and circuit shorts without unsoldering parts. The DP-1 digital pulser is a pocket-sized pulse generator used to stimulate logic circuits with either a single pulse or continuous pulse train. The LM-2A simultaneously displays the static and dynamic states of 16 logic inputs.

Each kit contains a logic probe to find pulses even too fast for an oscilloscope. The LTC-6 kit contains the LP-1 10-MHz probe; the LTC-7 kit contains the LP-3 35-MHz probe; and the LTC-8 kit contains the LP-5 100-MHz probe.

The price of the test kits range from $299 to $379.

Global Specialties
70 Fulton Terr. • New Haven, CT 06512
(203) 624-3103 • Fax: (203) 468-0060 Reader Service #504

DSP OPERATING ENVIRONMENT

An IBM PC-based operating environment, designed to develop or study Digital Signal Processing (DSP) algorithms, is available from BittWare Research Systems. **DSP Headquarters (DspHq)** allows an algorithm designer to write functions to operate on a block of data and it will supply the test data, handle file and memory management, and generate graphics. DspHq was designed to be an open architecture to allow users to integrate included functions, popular function libraries, and their own routines written in "C" or Pascal. Algorithms pass and share common data structures and can be calculated by PC or downloaded to a signal processor. DspHq includes a menu interface, command interpreter, batch command processor, file and memory management, and screen graphics with hardcopy support for dot-matrix, laser, PostScript, and HPGL devices.

The batch command language allows users to create and simulate an entire DSP system, complete with user-input, breakpoints, and single-step. Source code examples are provided for the popular "MathPak87" and "Numerical Recipes in C" analysis routines. Interfaces are currently available for most signal processing boards based on the AT&T DSP32/DSP32C devices, and others will be available soon.

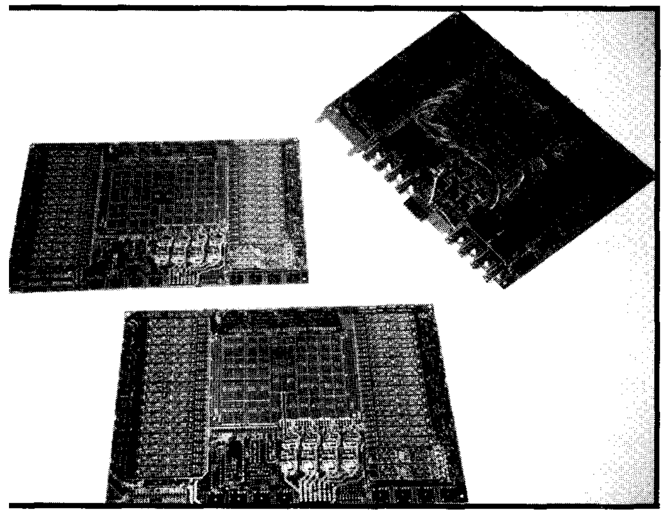
Other features include context-sensitive help, cross referenced for ease of use; pull-down menus; detailed parameter setup; up to eight display windows; and color output device support.

An introductory version is available for $250. The full release, with additional hardware and software interfaces sells for $495.

The demo disk can also be downloaded from the DspHq bulletin board at (301) 8383205. Modem parameters are 300/1200/2400-N-8-1. The BBS is devoted to Digital Signal Processing, DspHq, and other related topics.

BittWare Research Systems
400 East Pratt St., 8th Floor
Baltimore, MD 21202
(800) 848-0436
Fax: (301) 879-4465

Reader Service #505



UNIVERSAL ANALOG TEST INTERFACE BOARD

A universal analog interface that is controlled by a single 24-bit TTL digital I/O has been announced by Latel Engineering. The **FB-1000 Analog Test Interface Family** board is an 8" x 12" printed circuit board designed to provide a convenient and flexible interface between a Device Under Test (DUT) and the test system controller or microcomputer. It features a variety of signal interface and control functions and includes locations for 64 relays, two analog multiplexers, eight op-amps, eight voltage regulators, 16 latched digital output lines, and 16 latchable input lines. For custom circuit applications, a breadboard area is provided around the DUT that can accommodate up to 35 16-pin DIP components.

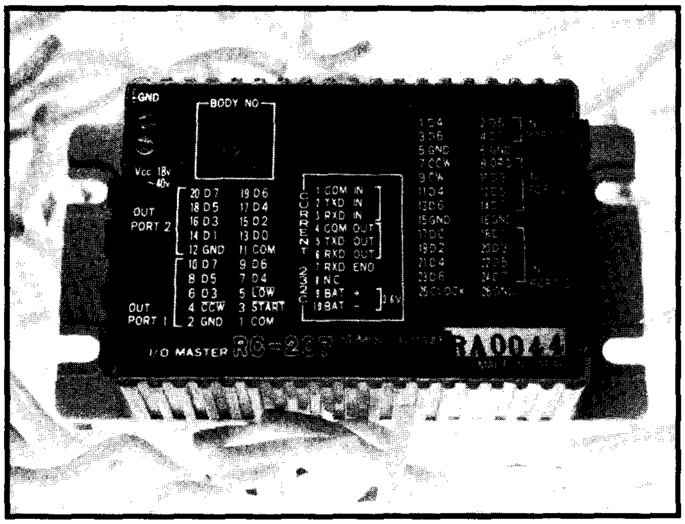
The FB-1000 uses a 24-bit control bus, made up of three 8-bit ports. Several manufacturers make 24-bit I/O cards for the IBM PC and compatibles, and sample IBM BASICA subroutine drivers to control the FB-1000 from two of these products are provided. Any 24-bit digital I/O controller that is TTL compatible can be used.

The FB-1000 package includes a parts list, schematic, board layout, instruction manual, software drivers, sample programs for the IBM PC and compatible computers, and the FB-1000 printed circuit board. The package, in quantities of one to three, sells for $250, dropping to $150 for quantities of 10 and above.

Latel Engineering
P.O. Box 391
Topsfield, MA 01983
(508) 887-9506

Reader Service #506

ADVANCED STEPPER MOTOR COUNTER/ CONTROLLER



A new advanced counter/controller has all the necessary commands and features to do complete motion and I/O control in one small package. The **I/O** Master **RC-**207 from Semix has a **high-**speed counter, I/O, and an increased command set that facilitates its use for distributed stepper motor control via a computer or as a stand-alone system.

The **110-kHz** counter and 1100 points of memory allow the RC-207 to do complex multiple-axis control. Up to twenty I/O Masters can be daisy-chained with just a three-line cable to a host computer located up to 2500 feet away. Each I/O Master can control two stepper motor drivers alternately and 20 inputs/l6 outputs; thus, centralized control and monitoring of as many as forty motors can be done from one computer. Programming can be done in a simple language such as BASIC or C.

The RC-207 can be configured for stand-alone control as well. By writing the program into each I/O Master's EEPROM, the units can be used without a host computer, except at initial programmmg.

The RC-207 features 8K bytes of EEPROM and 8K bytes of SRAM, as well as many commands to allow the process to be easily manipulated and changed. Some of these include: selectable baud rate, echo-back capability, jump routines, flag settings, counters, a timer, and stall detection. The unit's compact (2.2" x 4.1" x 1.1") size and rugged casing, which is noise and EM1 shielded, allows it to be placed in small places near mechanical components, reducing complex wiring and the usual noise generation. Price was not available at press time.

Semix, Inc.
4160 Technology Dr.
Fremont, CA 94538
(415) 659-8800
Fax: (415) 659-8444

Reader Service #507

ONDI— The ON-line Device Interface

Part 2

John Dybowski

Software for Remote PC Control

Having developed the hardware for a flexible remote computer control system, we will now investigate how the IBM keyboards work, and take a look at the prototype software for a simple remote control application. Our system configuration will consist of a computer, ONDI, and a Hayes-compatible modem; ONDI and the modem will remain powered up at all times.

THE FUNCTIONS OF A REMOTE CONTROL DEVICE

We won't get too fancy but we'll try to touch on some of the more important elements. To get started we must have the capability to control and monitor the computer power. Keyboard emulation for both PC/XT- and AT-style keyboards will be needed. The modem will typically be required to auto-answer an incoming call, which we will want to enable or disable under software control. This will be accomplished serially via the Hayes AT command set by either setting the modem to answer on the first ring or by issuing the reset command which will return the modem to its switch-selected settings.

To ensure that the modem is always configured to the desired state, we will also monitor DSR and transmit the appropriate sequence on a transition of DSR going from off to on which would indicate either the connection or powering up of the modem. For ease of implementation, the modem's result codes will be disregarded and the modem's CD line will be used to find the state of carrier.

To simplify the user interface, the operational state of ONDI will be controlled by a single three-position toggle switch which will enable or disable auto answer, or put the system into a local mode of operation. ONDI will have the capability of being serially accessed either locally **or** remotely, where remote access will be used for control purposes and local access for configuration, check out, and downloading. Interactive communications will be carried out using a command line format which will handle buffered input with echo, and a transparent mode of operation will release the communications lines to the modem/computer. An escape sequence will be used to return to the command line.

A timeout facility monitors the modem line for periods of inactivity. Abort times will be set at two minutes during log on and five minutes for normal remote operation and will terminate the session on expiration.

With the power to access the resources of a computer remotely, there is a potential for problems, so password access protection will also go on our list. Finally we will twiddle some LEDs on ONDI to indicate these various states and modes of operation.

STATES AND MODES

Having defined the desired features, it is now possible to outline the operational environment in which ONDI will operate. ONDI runs in one of three primary states: idle, local, or remote. When either local or remote, ONDI can function in a transparent or command line mode. Command line mode puts the user in direct communication with ONDI. Transparent mode allows direct communications between the local and remote computer. Priority is given to the local interface so ONDI can transfer directly from remote to local operation if the switch is thrown to the local position. The following rundown briefly describes the major operational states and modes that ONDI can assume:

Idle: The idle state is entered on power-on following system initialization and is the point of return following the termination of most other states. ONDI waits here for something better to do.

Remote state, log-on mode: This modeallowstheuserattheremotesite to access ONDI in order to satisfy the log-on requirements. If successful, ONDI proceeds to the remote command line mode.

Remote state, command line mode: In this mode the remote computer is in direct communication with ONDI and has access to the remote command set.

Remote state, transparent mode: The computer's COM port is connected to the modem. ONDI monitors the modem transmit line. If the escape sequence is detected, control returns to the remote command line mode.

Remote state, CTTY mode: Like the transparent mode, except that ONDI issues the DOS MODE and CTTY commands to the local computer via simulated keyboard input prior to entry. On exit ONDI CTTYs back to the console via the COM port.

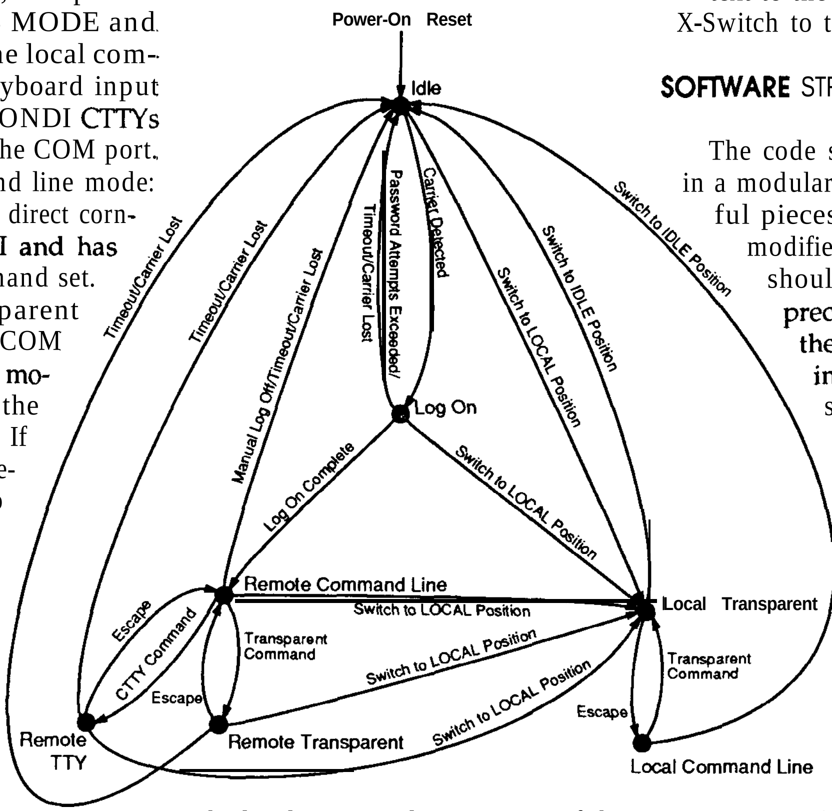
Local state, command line mode: The local computer is in direct communication with ONDI and has access to the local command set.

Local state, transparent mode: The computer's COM port is connected to the modem. ONDI monitors the computer transmit line. If the escape sequence is detected, control returns to the local state/command line mode. This is the point of entry when the mode switch is set to local operation.

NOW, THE COMMAND SETS

Where possible, to reduce the amount of unique command mnemonics, the practice of using the command without an argument is used to return the function's status. In entering the commands, upper or lower case is OK. The remote command set is as follows:

- H-Display a list of these commands on the remote console display.
- S-Power status; returns ON/OFF indicating the computer power status.
- S+ -Turn computer power on. Report if computer logic power is not detected within five seconds, delay for boot-up (as programmed via the local command line mode).
- S- -Turn computer power off.
- K<text>—Send <text> as simulated keyboard data.
- C-CTTY to COM port (as selected via the local command line mode), and switch to transparent operation.
- X-Switch to transparent operation.
- L<text>—Line check, retransmit <text> to the remote computer.
- A-Abort, hang up line.



The local command set consists of the following:

- H-Display a list of these commands on the local console display.
- M-Display computer mode: AT or PC/XT.
- Ma-Select AT mode.
- Mx-Select PC/XT mode.
- B-Display boot delay time.
- Bnn-Set boot delay time, 0 to 99 seconds. The boot delay time is the amount of time ONDI waits after executing the S+ command in remote mode. This is to prevent accidentally injecting simulated keyboard data before the keyboard's power-on transactions have been completed.
- C-Display selected COM port.
- Cl-Select COM port 1.
- C2—Select COM port 2. This is the COM port used when the Ccommand is processed in remote mode.
- K<text>—Send <text> as keyboard data.
- P-Display password
- Pc"text">-Set password.
- P""—Deletes the password. The password is case sensitive.
- L<text>—Line check, retransmit text to the local computer.
- X-Switch to transparent operation.

SOFTWARE STRUCTURE

The code structure is segmented in a modular fashion such that useful pieces can be extracted or modified for your own use. It should be evident from the preceding discussions that the main application runs in various loops corresponding to the various statesand modes. Also, it should be noted that, for the most part, the required lower-level support functions remain consistent as we traverse the various process loops.

With the above in mind, the structure adopted for the mainline processing is composed of two parts: the main (MAIN) application driver, which handles the specifics of the moment, and the low-level resource handler (IDLE) that's responsible for performing the grunt work.

IDLE is a callable routine that is the interface between the application and the hardware and background interrupt functions. The functions of IDLE include monitoring and debouncing the toggle switch and modem CD and DSR signals (returning only when a change in state occurs), and relaying the SIO-related information such as receive complete, receive timeout, receive error, and escape detect. These events are indicated by returning result codes to the calling routine in the accumulator. The benefit of this approach is that the application software becomes much simpler to code and comprehend since the details of handling the data input are centrally processed in a routine that, once we have working, we will not be apt to disturb. Typically the application calls IDLE whenever it has completed its specific function such as interpreting a command or transmitting a string of data. On return, the application has the option of either processing or ignoring the return code as appropriate. A quick look at this arrangement confirms the premise that a typical program executes five percent of its code 95 percent of the time.

IDLE's return codes are:

1-Received text available
2-Receive error (command line overflow)
3-Escape sequence received
Q-Receive timeout
5—Enable local operation
6—Disable local operation
7—CD went on
&CD went off
9—DSR went on
IO-DSR went off
11-Enable auto answer
12-Disable auto answer

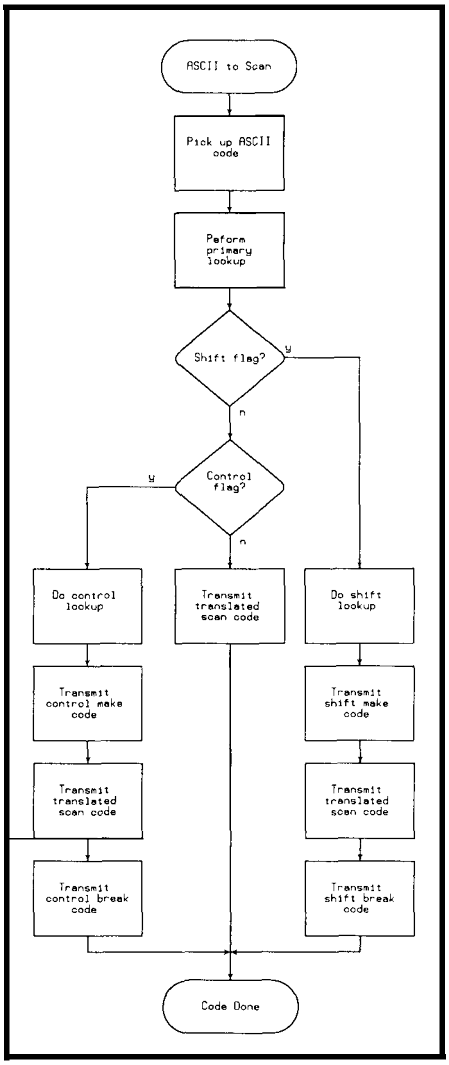
THE KEYBOARD ACCORDINGTO IBM

The keyboard as implemented by IBM for its PC/XT and AT computers performs physical key scanning using an on-board processor that detects when a key is pressed or released and signals the computer by sending a make or break scan code. Keyboard scan codes are assigned by numbering the physical keys on the original PC keyboard from left to right, top to bottom. The computer's BIOS converts the unique key codes to ASCII representation (where possible) for use by the computer. Keys that have no corresponding ASCII symbol areassigned a value of ASCII null which indicates that the next byte should be interpreted as a scan code and not an ASCII code. The keyboard communicates to the computer over an interface that consistsof a data and clock linedriven by open-collector devices that are pulled up with resistors at each end.

The PC/XT keyboard computer interface is implemented in hardware using a shift register where the data bits are sequentially shifted in on the falling edge of each clock transition. When an entire character has been assembled, the receive logic issues an interrupt and simultaneously pulls the clock line low until the received character has been processed. This is an indication to the keyboard that it now must not transmit. An idle keyboard releases the clock line allowing the pull-up resistors to hold it at a positive state and asserts the data line to a logic low; therefore, the interface is capable of unidirectional traffic only. The PC/XT keyboard is capable of encoding 128 unique key codes. A byte with a value of 0 to 127 is considered a make code. Break codes are formed by adding hex 80 to the make codes. For example, key 1 produces scan code Olh on make and 81h on break. The bit format for data transmission consists of a "1" start bit and eight data bits.

The AT keyboard uses a bidirectional serial interface to carry signals between the keyboard and the computer. As in the PC/XT keyboard, all keys are classified as make/break, however in order to increase the total number of keys that can be encoded, the method of indicating break codes is different. When a key is released, the break code consists of hex FO followed by its make code. Since the interface is used for bidirectional communication, the data and clock lines are both released and passively pulled up when the keyboard is idle. This permits either the keyboard or the computer to take control of the interface. Data transmission to and from the keyboard consists of 11-bit data streams composed of a "0" start bit, eight data bits, odd parity, and a "1" stop bit. Unlike the simple hardware interface of the PC/XT, the AT uses a dedicated microcontroller to communicate with the keyboard. This arrangement incorporates a communication protocol that provides for errordetection, timeoutabort, retransmission, and recovery from line contention. A command set is also supported both to and from the keyboard. Fortunately, experience shows that, for the most part, these elements can be ignored and reliable communications can be attained by simply clocking the key codes into the computer.

The basic procedure for translating ASCII data to key codes is the same regardless of which style keyboard we use. The method basically consists of performing a table lookup using the ASCII character as an index. The table returns either the translated scan code, a flag that indicates that a secondary lookup must be performed using the shift table, or a flag that indicates that the control table must be consulted. Once the final scan code is determined, the sequence is assembled and transmitted to the computer. For unshifted characters, only one scan code is transmitted, but in the case of shifted or control characters, the scan code must be framed with a shift or control make code and a shift or control break code as appropriate. Note the performance degradation when transmitting strings of shifted characters, particularly in the AT emulation routines, since for every shifted character, three extra bytes must be transmitted. It will be left as an exercise to the reader to consider how to keep track of the shift mode so we won't have to frameeachand every shifted character with the shift sequence. Note that the only break codes that are transmitted are for the shift and control keys. Our keyboard emulation routines have the capability to transmit the source string from program memory or internal data memory. Finally, bear in mind that we're working under the assumption that the keyboard isinactiveand that we're operating from a known state, otherwise we would have to constantly monitor the keyboard traffic for codes such as ALT make, shift make, control make, and the toggling of caps lock.



SERIAL COMMUNICATIONS

Communications for the system are fixed at 1200 bps, no parity, and one stop bit. Due to the nature of the discrete memory areas in the DS2250, the transmitter can be set up to transmit from program memory, external data memory, or internal data memory. Although the transmitter is fully interrupt driven, it is used in a polled mode in this application. The receive interrupt handler implements the buffered command line where the backspace character allows editing of commands. Received character echoing is handled from the receive interrupt and may be enabled or disabled. Receive timeout and escape sequence detectionarehandled withalittle help from the timer interrupt routine.

The escape sequence is defined as at least one second of silence followed by three "minus" characters. The receive abort timer is implemented as a softwareretriggerableone-shotthatis rearmed every time a character is received. The receiver signals the foreground when a return is received or on buffer overflow indicating either normal completion or an error.

SYSTEM TIMEBASE

All system timing is referenced to the timer interrupt routine that executes every 25 milliseconds. In addition to handling the escape sequence guard time timer and receive abort timer, a general-purpose timer is supported for counting delay times from one second to five minutes. A sync flag is also set every time the timer interrupt is invoked.

PUSHING THE CODE AROUND

Although we could use any standard communications program with file **manipulation capabilities** to transfer our hex file to ONDI, this would require a multistep process to configure the DS2250, load the program, and verify the program. A better solution is my LOADER utility. Not only does it build a DS2250 configuration file for setting up the chip, but the program will accept a command file either on the command line or interactively that will not only direct LOADER to execute the required programming steps, but will also configure LOADER'S operational communication parameters. The process of downloading to ONDI simply consists of setting the load switch to the load position and invoking LOADER. Once done, we set the load switch to the **run position** and ONDI immediately begins executing the program. ***[Editor's Note:*** *Soffware for this article is availablefrom the Circuit Cellar BBSand on Software On Disk #18. For downloading and* ordering informa*tion, see page 92.1*

THE EXPANDED CONTROL DEVICE

After test driving the prototype code, it is apparent the device can in fact operate a computer remotely. Still, this rudimentary implementation requires human intervention from the remote computer and is quite limited. Our DS2250, with its inherent intelligence, however, can be programmed to function independently. First, we might implement the capability of recording key code sequences directly from the keyboard so that we could later play them back to the computer. Next, wecould provide for stand-alone initiation of sequences, where ONDI would operate the computer using predefined sequences in response to external events. With the addition of a communication protocol, we could relegate remote communications to run between unattended computers.

Consider a secure modem application. The remote calls in and leaves a password. ONDI hangs up, powers up the computer, and starts a program inputting the password on the command line. Now, the computer takes over and searches its database, logs the transaction, and, if the password has a corresponding telephone number, it dials it directly. This turns out to be an almost trivial exercise, mainly because the computer is used to compute and the controller to control. Just a simple matter of software. Stay **tuned.**

My *appreciation goes out to Dan Burkefor his* participation in this project.

John Dybowskihas been involved in thedesign and manufacture of *hardwareand software for industrialdatacollectionand communications equipment. His crowning achievements are his daughter Ondi and his son John.*

IRS

400 **Very** Useful
401 Moderately Useful
402 Not Useful

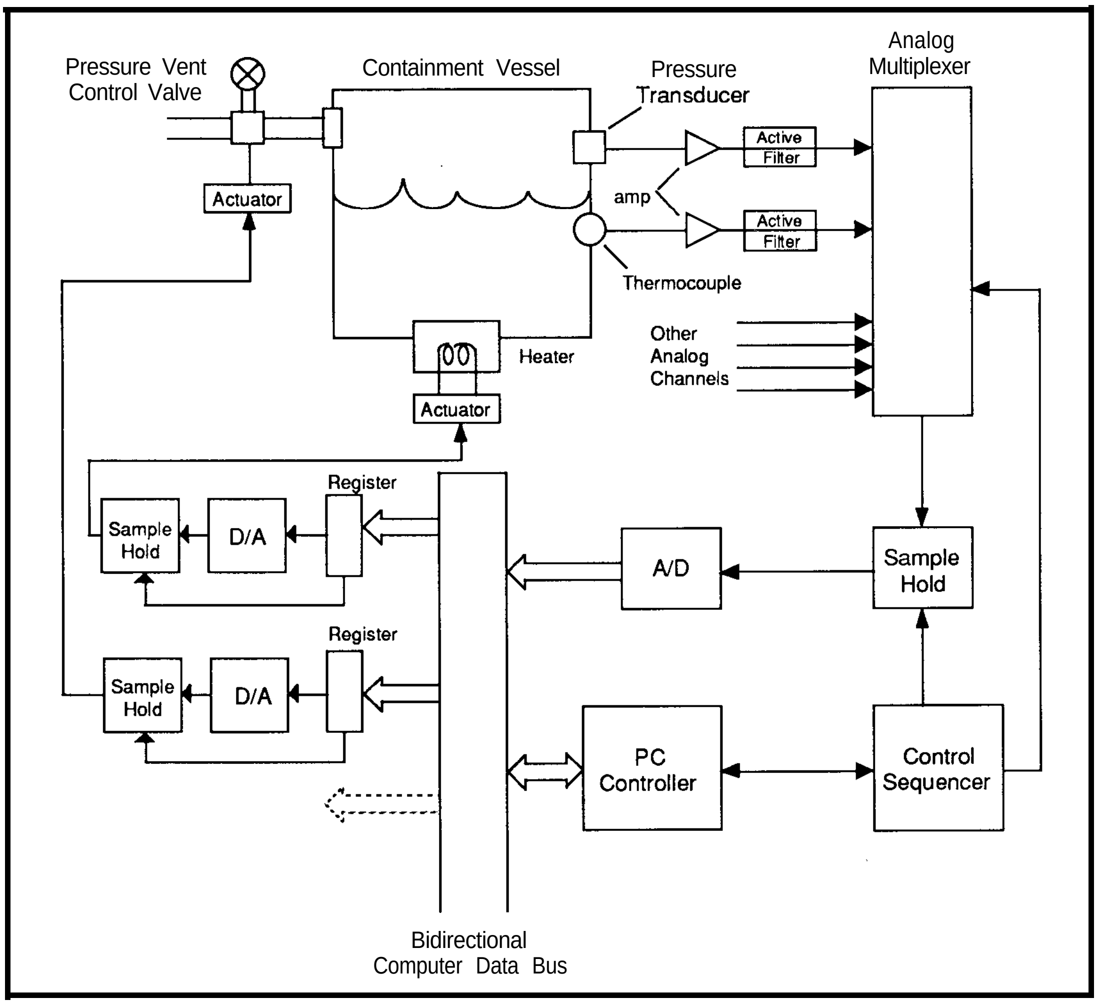
FEATURE ARTICLE

Chris Ciarcia

Principles of PC-based Data Acquisition and Control Systems

I could start this article with some sweeping statement like "PC-based data acquisition and control systems provide the greatest versatility and flexibility, etc., etc.," but most of you are already sold on that idea. If you've spent any time reading past issues of this magazine then you appreciate how many different things your PC can do. Instead, I'd like to discuss the basic principles behind the application instrumentation used in those articles. In other words, I want to discuss how your PC is able to interact so effectively with the real world and what pitfalls you may encounter as you try to control your environment,

Figure 1 -Process DAAC System. Physical parameters and process characteristics are derived from pressure (transducer) and temperature (thermocouple) measurements which are amplified and digitized and then fed to a PC for status monitoring and decision input. The PC then analyzes the process response, calculates error and generates a series of control sequences which are fed back to the process for control manipulation.



We can all appreciate that the world around us very rarely lends itself to simple understanding, evaluation, or control, especially if we attempt our control using a computer system. Most real-world physical parameters are analog and require some form of interface to the world of artificial computation and control.

DATA ACQUISITION AND CONTROL

The basic interfacing modules which are employed in most real-world interactive systems are the analog-to-digital (A/D) and digital-to-analog (D/A) data converters. The A/D converter is a circuit which converts an analog (continuous) voltage or current into an output digital code. And conversely, the D/A converter converts a digital code word into an output analog voltage or current. Of course, that's not all there is to a real-world interface. In reality DAAC systems are created using one or more of the following components:

- A/D and D/A converters
- transducers
- sensors
- amplifiers
- filters
- analog multiplexers
- sample-holds
- digital control sequencers
- PC computer controller and I/O bus
- actuators

To see how the above **components** can be combined to create a workable computerized feedback control system, consider the diagram shown in Figure 1. Here, measurements of "some heating process" within a pressurized vessel are used to monitor and generate feedback control sequences which will optimize the desired process. The primary inputs to this DAAC system are the measurements of (analog) physical parameters such as temperature, pressure, velocity, acceleration, or displacement.

To gain an elementary understanding into the design configuration of the needed elements within the DAAC system shown, let's trace the

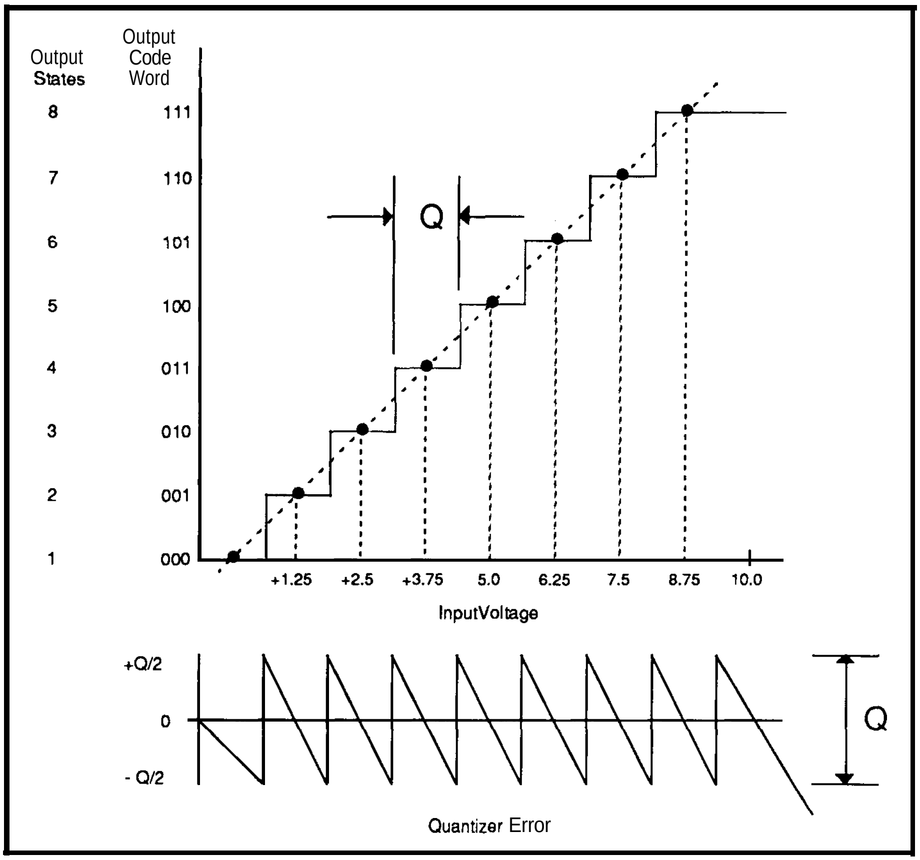


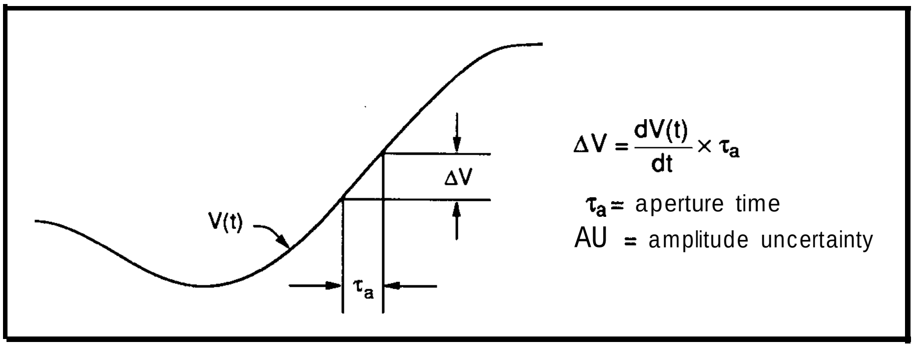
Figure 2-Due ***to the stair-step nature of the digitization process, there is always a small amount of quantization error associated with the digitized signal.***

pressure measurement from the containment vessel's pressure transducer through the interface, to the computer. Then we'll follow the computer's response back through the control interface, to the actuator located at the pressure vent control valve, where modifications in the processes' status will be made.

Let's assume that the pressure (or force per unit area) within the containment vessel is measured using a standard pressure transducer. Here, an elastically deformable sensing element responds to an omnidirectional fluid/gas pressure. The deformation within this element, usually the compression of some sort of bellows or convoluted diaphragm, is then measured and an electrical signal is generated which varies as a function of that deformation displacement. This analog signal is then fed through an amplifiertoboost theamplitudeofthe transducer output signal to a reasonable level for down-line processing. The typical transducer output usually rangesover the microvolt-to-millivolt level and may be a high-impedance signal, a differential signal with common-mode noise, a current output, a signal superimposed on a high voltage, or a combination of these. It is usually amplified to a range of 1–10 volts.

Theamplifier output is channeled through a low-pass filter which reduces the undesirable high-frequency components (noise) within the signal. Sometimes the signal is then passed through a nonlinear analog function circuit which may modify the signal by squaring, multiplying, dividing, doing RMS or log conversion, or linearizing it.

The processed signal is introduced to an analog multiplexer which sequentially switches between a number of different analog input channels. Each input is connected to the output of the multiplexer for a set time interval during which the sample-and-hold module receives the signal voltage for sampling while the A/D converter converts that value into a digital form. This resultant digital word is then



gure 3—*Any* analog to *digitalconversion takes a finite amount of time during which the in,put signal may change,*

dumped onto the DAAC system's bidirectional data bus for access by the PC controller.

This, of course, is not the only approach to data acquisition. In an often-used alternative, the measured signal is amplified and converted into digital form at the transducer: The output is then transmitted to the PC through a serial port. However, implementation of this technique requires that the data be converted to parallel form and then multiplexed onto the data bus.

In the feedback control portion of the DAAC interface, for all intentsand purposes, we do the inverse of the input data acquisition process. The computer, based on analysis of its input, determines the necessary output control functions required to elicit a specified response within the "defined process" according to its operational guidelines. These control outputs are digital in form and must be converted to analog form to drive the process. This conversion is accomplished by employing a D/A converter, usually coupled to a storage buffer which stores the digital word until the next update. These buffers are under the control of the sequencer. The output of the D/A converter is used to drive actuators which directly control the various parameters.

QUANTIZATION AND SAMPLING THEORY

Many of the commercially available A/D and or D/A boards may combine several of these individual functions. But understanding the basic concepts behind each of the above described components is of extreme importance when designing and configuring your DAAC system, or for when you use an "integrated" data acquisition board. Your original analog data is manipulated and altered by these devices. So how you quantize, sample and code your measured data will in fact determine the quality of your measurement results. Here, quantization refers to the process of transforming a continuous analog signal into a set of discrete output states which are then coded into a discrete digital word which represents each output state, **during a** specified sampling time interval.

THE QUANTIZER TRANSFER FUNCTION

Quantizing an input analog signal divides the signal into discrete levels or states whose size or range determine the overall resolution of the A/D process. Typically this conversion is characterized by a nonlinear transfer function like that shown in Figure 2. This figure represents the quantizer for an ideal 3-bit system with eight output states, like that used in a 3-bit A/D converter. These eight states are referenced by a sequence of binary **numbers** which range from000 to 111 over an analog input range of O-10 volts.

The absolute resolution of the quantizer circuit is specified by the number of output states defined in bits, or, in terms of a binary-coded quantizer power, 2^n, where n is that number of bits. Thus, an 8-bit quantizer has 256 output states and an 24-bit quantizer has 1,677,220 output states. Most applications call for a 2^{12}, or 4096, output format.

In our example diagram there are 2^n-1 analog decision points or threshold levels within the transfer function (at +0.625, +1.875, +3.125, +4.375, +5.625, +6.875, and +8.125). These decision points are precisely set halfway between the code word center points (+1.25, +2.50, etc.) in order to divide the analog voltage range into correct quantized values. So, when an analog signal level is submitted, the quantizer circuit determines what threshold level has been obtained and assigns the binary code corresponding to it. Keep in mind, for anyone output code word there exists a small range of voltage differential about its center point where an input voltage can be assigned the same code. This range between any two adjacent decision points is known as the analog quantization size, quantum Q. For our example (shown in Figure 2), the quantum is Q=1.25 V. In general it can be determined by dividing the full-scale analog range by the number of output states. Q therefore represents the smallest analog difference which can be resolved or differentiated by the A/D quantizer.

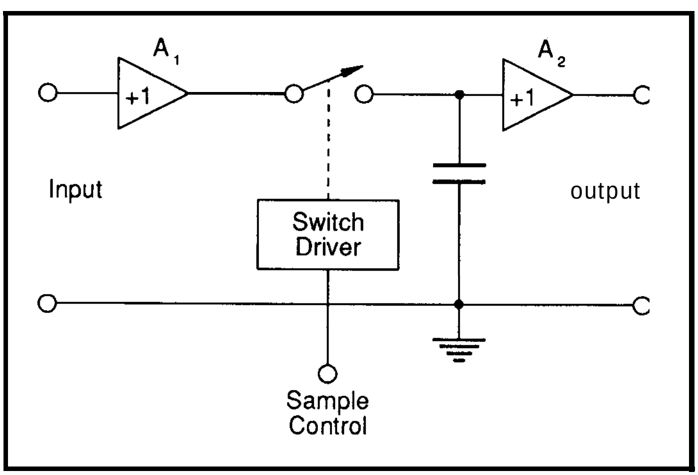


Figure 4-h *order to keep the input signal steady during the conversion, a sample and hold circuit is typically used.*

A sawtooth error function for the quantizer can be derived by running the input through its entire range while taking the difference between the output and input. This is the irreducible error that results from the **quantization** process. It can only be minimized by increasing the number of output states (the absolute resolution). Typically, for a given analog input level, the output will vary anywhere from 0 to $\pm Q/2$, with the error being zero only at points with analog values that correspond to the code center points. This variation is defined as the quantization uncertainty or noise. As a result, the **quantizer** output can be thought of as a combination of the analog in and the Q noise, where this noise has a peak-to-peak value of Q and **an average** of zero. The **RMS** value can be computed from the triangular waveshape. It is useful in analysis and is usually Q/3.464.

SAMPLING THEORY

No process is instantaneous! Like most things, the A/D converter requires a small but significant amount of time to complete the quantization and coding steps. This "active" time depends on several things, such as the overall resolution, theconversion technique, and the component processing speed. This aperture time refers to the time uncertainty or "the time window" during which a measurement is made. And as such it results in an amplitude uncertainty if the signal changes during this interval.

To understand this better, consider the diagram **shown** in Figure 3. Here, the input to the A/D converter changes by *AV*, the maximum error due to the signal change during the conversion time window, τ_a. The error in this measurement **canbe** estimated in two ways: as an amplitude error or as a time error, which are related to each other by:

$$\Delta V = \tau_a \frac{dv(t)}{dt}$$

where $dv(t)/dt$ is the rate of change with time of the input signal. If a sinusoidal input signal is employed, the maximum rate of change occurs at the zero crossings of the waveform, and the amplitude of the error becomes,

$$AV = \tau_a \frac{d}{dt}(A \sin \omega t)_{t=0} = \tau_a A\omega$$

The resultant error as a fraction of the peak-to-peak full scale is

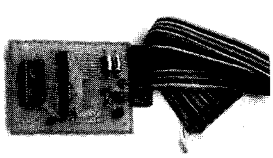
$$e = \frac{\Delta V}{2A} = \pi f \tau_a$$

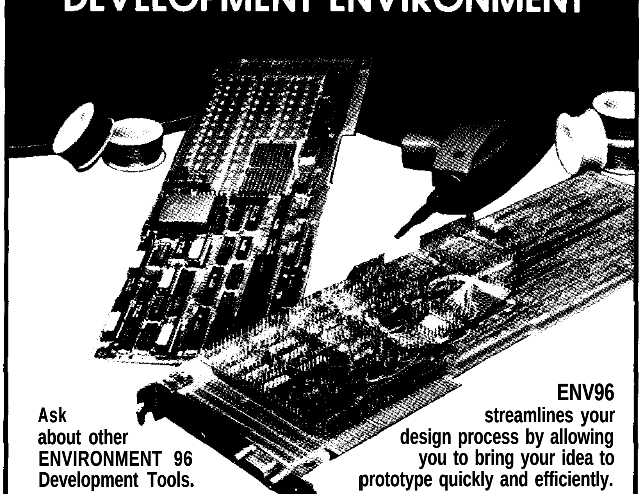
For example, the aperture time required to digitize a **10-kHz** signal to 12 bits of resolution can be estimated as,

$$\tau_a = \frac{e}{\pi f} = \frac{0.0002441}{3.14159 \times (10.0 \times 10^3)} = 7.7 \times 10^{-9}$$

that is, 7.7 nanoseconds. With numbers like these, one begins to understand the difficulty converting signals in the tens of gigahertz range, where some of the latest scopes are now employed.

Also affecting the overall aperture time is the sample-and-hold time.





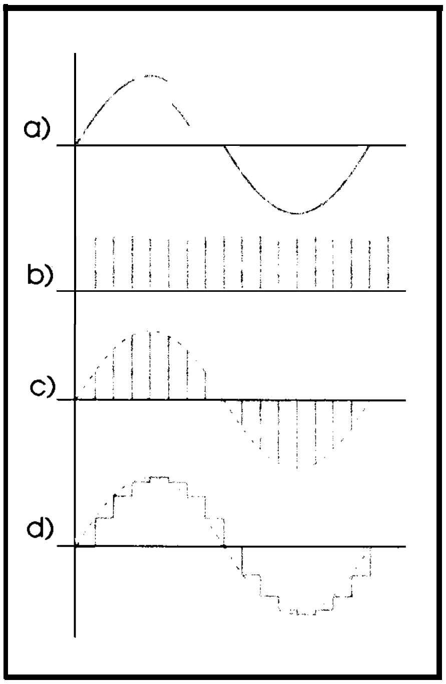


Figure 5—*The digitization process involves sampling the original signal at fixed intervals. The samples are then used to reconstruct the original.*

It is derived from the device that samples the input signal voltage and stores it on a capacitor for the time required by the A/D quantizer to do its thing. This means that the aperture time of the A/D converter is often reduced by the "typically" shorter time of the sample-and-hold circuit, which is a function of its bandwidth and switching time.

A sample-and-hold is simply a voltage memory device in which an input voltage is stored on a high-quality capacitor. A popular version of this circuit is shown in Figure 4. Here, *A,* is a high-impedance input buffer amplifier, such as an analog multiplexer. *A,* is the output amplifier which buffers the voltage on the hold capacitor C, which has low leakage and low dielectric absorption characteristics (typically MOS type). S is an electronicswitch,usuallyaFET, which is rapidly switched on or off by a driver circuit which interfaces with TTL inputs.

The sampleand-hold works in two modes: sampling (or tracking) when S is closed, and holding when the switch is open. When used with an A/D converter, the device is kept in a hold mode until a new input signal is required. This is the case for a sample-and-hold used in a data acquisition system following a multi-

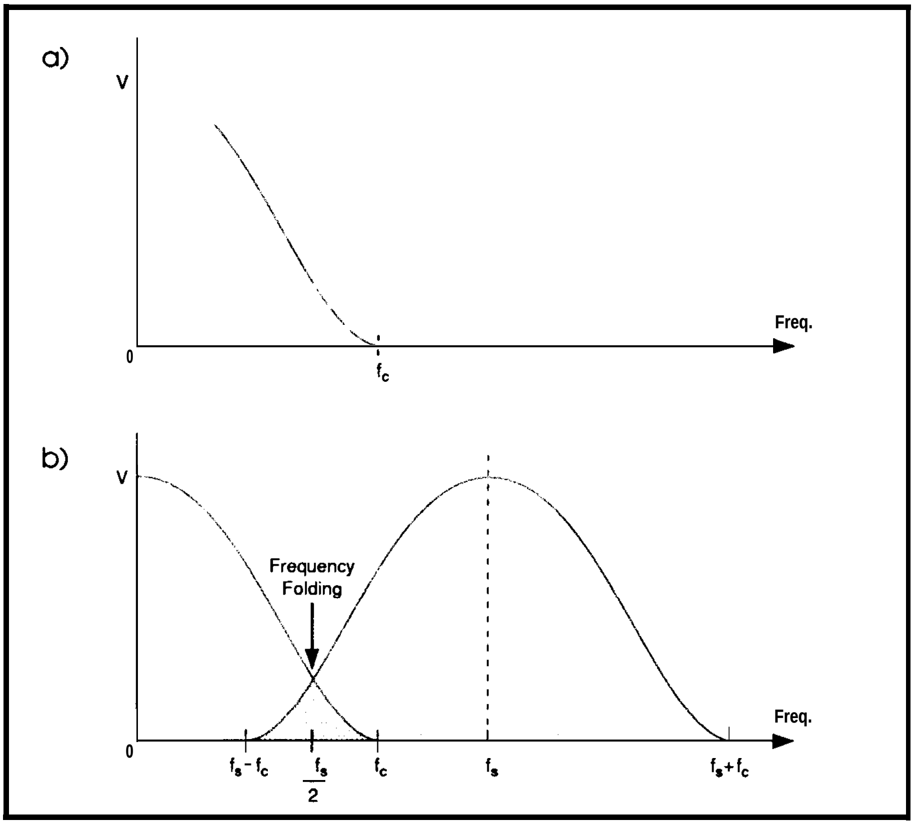


Figure 6-When *the frequency spectrum shown in (a) is sampled andreconstructed at a rate less than twice f_c, frequency folding or aliasing results (b).*

plexer (see Figure 1). While with an D/A converter, it is used as a deglitcher. Here, the device is used to continuously track the input signal, then switched into the hold mode only at specified times.

THE SAMPLING THEOREM

In most DAAC systems, analog signals are sampled on a periodic basis, similar to that demonstrated in Figure 5. Here the sinusoidal input signal (5a) is sampled on a periodic basis, determined by a train of sampling pulses (5b) which represent a fast-acting switch that connects the signal to the sample-and-hold for a very short period of time, then disconnects it for the remainder of the sampling period. The result of this operation is identical to multiplying the analog pulse by a train of sampling delta functions of unity amplitude (see Figure 5c). The amplitude of the original signal is **preserved** in the modulation of the pulse envelope with the amplitude of each pulse (sample) being stored between samples within the sample-and-hold. From this, a reasonable reconstruction of the original signal can be achieved (see Figure 5d).

Now the **important** question must be answered: "How much sampling must I do (what resolution) to retain most of my input information?"

Obviously, if I sample at a rate where little or no change takes place between samples, I won't lose much. Equally obvious is the fact that information is being lost if there is a large amount of change in signal amplitude between samples. So to get an answer, let's consider the Sampling Theorem, which states:

"If a continuous, bandwidth-limited signal contains no frequency components higher than f_c, then the original signal can be recovered without distortion if it is sampled at a rate of at least $2f_c$ samples per second."

The best way to understand this is to consider the diagrams in Figure 6. Here, Figure 6a shows the frequency spectrum of a continuous bandwidth-limited analog signal with frequency components out to f_c. When the signal is sampled at a rate of f_s, the modulation process shifts the original spectrum out to f_s, $2f_s$, $3f_s$, and so on, in addition to the one at the origin (a piece of this spectrum is in Figure 6b).

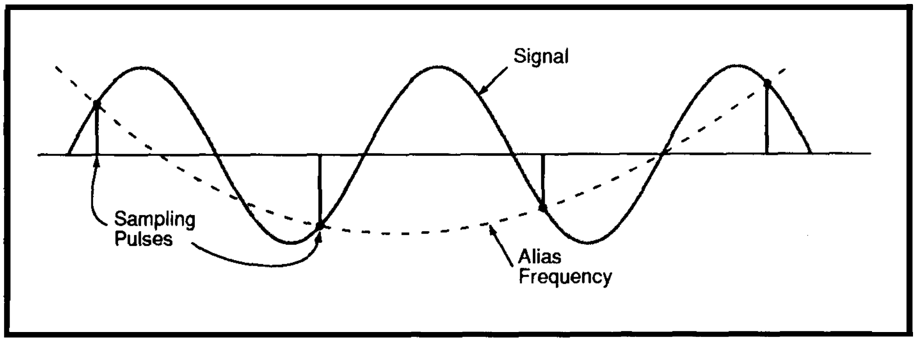


Figure 7-An inadequate sampling rate causes an alias frequency.

If the sampling frequency f_s is not high enough, then the part of the spectrum centered around f_s will fold over (frequency folding) into the origi-

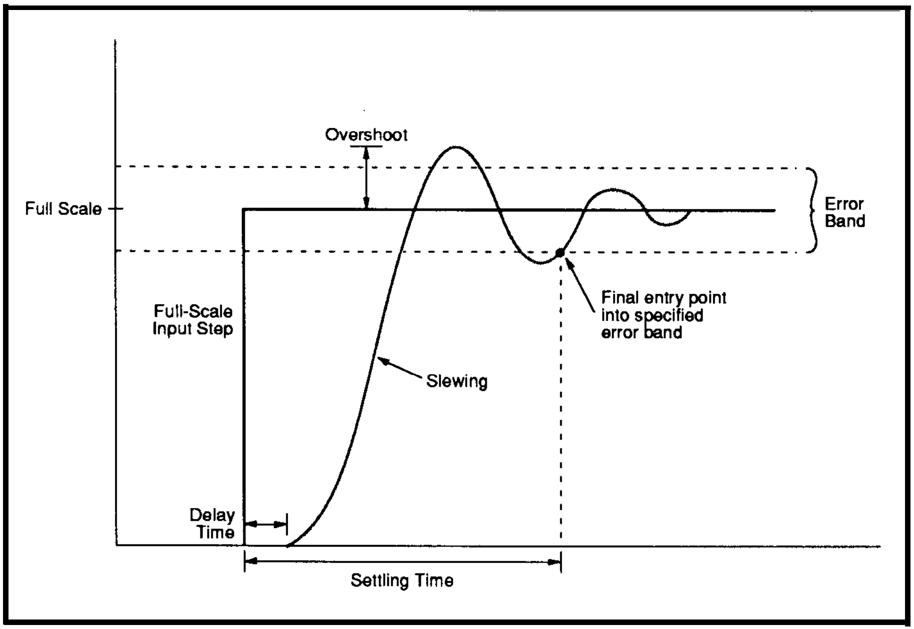


Figure 8—*The length of an amplifier's settling time depends on how long it takes the output to settle into the limits of the error band.*

nal signal spectrum. As a result, the folded part of the spectrum causes distortion in the recovered signal. Otherwise, if the sampling rate is increased to $f_s - f_c > f_c$, then the two spectra are separated and the original signal can be recovered without distortion. Therefore, frequency folding can be eliminated by either using a sufficiently fast sampling rate or by filtering the signal before sampling to limit its bandwidth to $f_s/2$.

Figure 7 demonstrates the effect of an inadequate sampling rate on the sinusoid in Figure 5. If a sampling rate less than twice per cycle is employed, an alias frequency in the recovered signal willresult. However, if, according to the Sampling Theorem, we usearategreaterthantwice a cycle, the original frequency will be preserved.

AMPLIFIERS AND FILTERS

I'm not going to say too much about amplifiers and filters within this article, since most of you have been repeatedly **exposed** to those devices. Suffice it to say that the front end of our DAAC system (shown in Figure 1) extracts a signal from a device that measures a physical parameter (our pressure transducer or thermocouple) and thenamplifiesand filters it. Both are critical to the system's overall performance. The amplifier may act to boost the signal amplitude, buffer the signal, convert a signal current into a voltage, or extract a differential from common-mode noise. A filter, typically a low-pass antialiasing filter, is often used to reduce man-made electrical interference noise, reduce electronic noise, and to limit the bandwidth of the analog signal to less than half the sampling frequency in order to eliminate frequency folding.

SETTLING TIME

An important aspect of any data acquisition system is called the settling time. That is, "the time elapsed from the application of a full-scale step input to a circuit to the time when the output has entered and remained within a specified error band around its final value."

Its importance lies in the fact that our DAAC system is made up of several analog operations which must be performed in a set sequence. As such, each operation must be accurately settled before the next operation can be initiated. For example, a buffer amplifierprecedinganA/Dconverter must have accurately settled before the converter can do its thing. An example of this settling time is shown in Figure 8. Here a full-step input which has been submitted to an amplifier is followed **by a** small time delay afterwhichtheamplifieroutputslews or changes its maximum rate resulting from internal amplifier currents which must change internal capacitances. Then, as the amplifier approaches its peak value, it may overshoot and then reverse and undershoot the full-scale value before finally entering and remaining within the specified error band. As seen within Figure 8, this settling time is measured from the start time to the final entry into the specified error band (typically ±0.01% to ±0.1% of the full-scale transition).

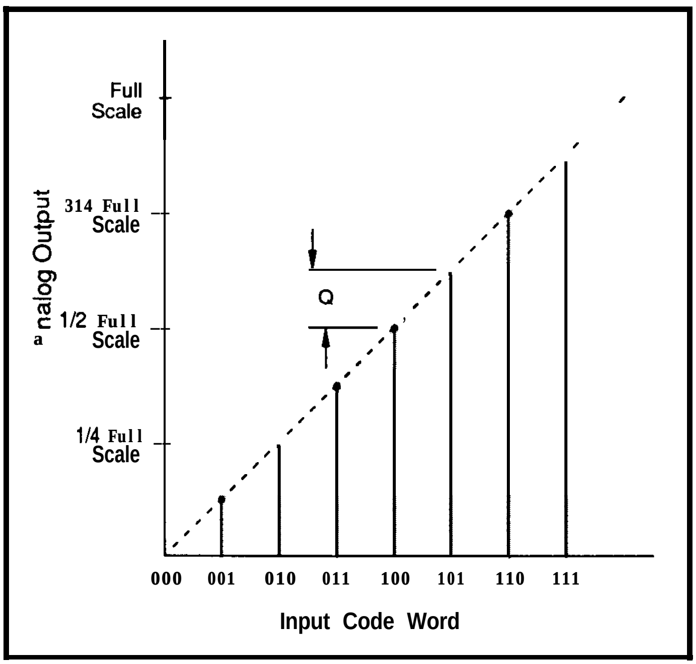


Figure 9-The transfer function of an ideal 3-bit DAC.

ANALOG **MULTIPLEXERS**

An analog multiplexer is the device which time-shares an A/D converter between a number of analog input channels. It is composed of an array of parallel switches connected to a common input line (with only a single feedthrough active at one

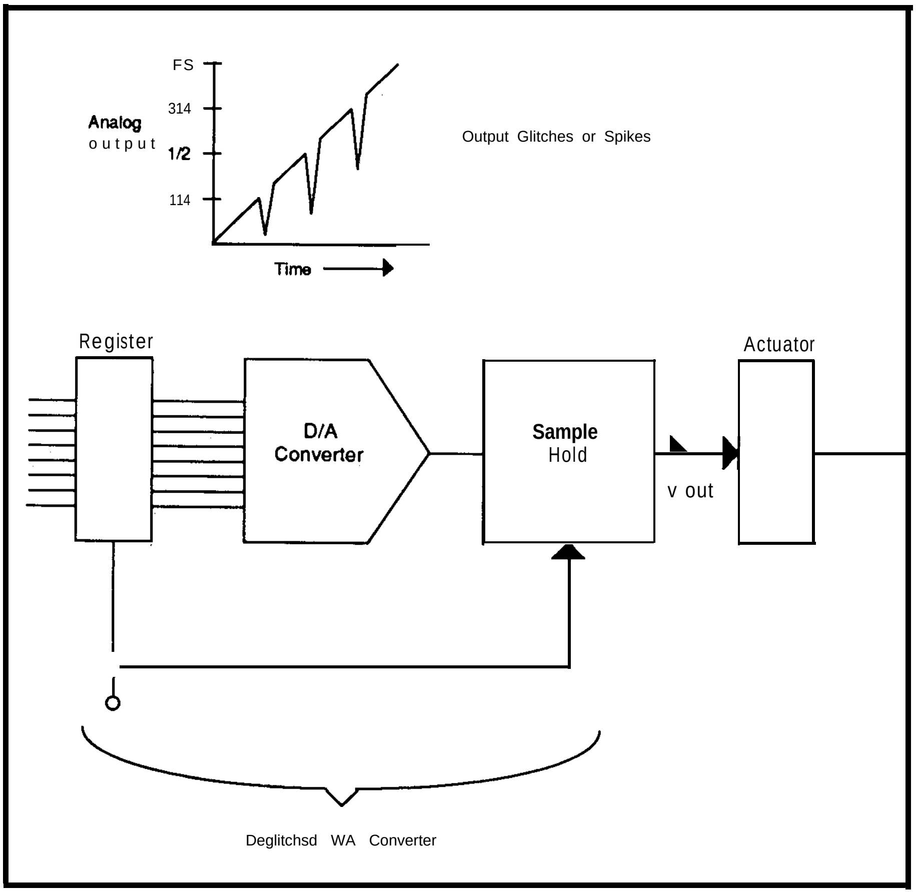


Figure 10—Output spikes (glitches) and a deglltched DAC-actuator system.

time), and a decoder-driver circuit which decodes a binary input word that specifies the appropriate switch setting. The decoder-driver interfaces with standard TTL logic.

Primary characteristics of the multiplexer, which you should keep track of, are:

transfer error-this is defined as the input-to-output error of the multiplexer with the source and load connected, expressed as a percent of input voltage.

break-before-make switch-a small time delay between the connection to the next channel and disconnection from the previous channel which assures that the two adjacent channels are never simultaneously connected together.

settling time-defined the same as in our amplifier example except that it is referenced from the time the channel is turned on.

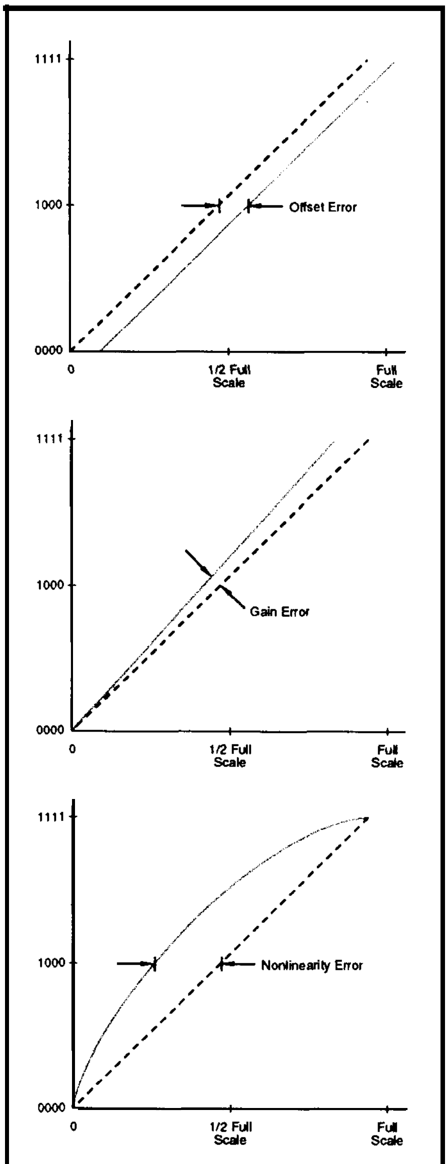
fhroughpuf rate-the highest rate atwhichamultiplexercanswitchfrom channel to channel with the output settling to its specified accuracy.

cross talk-the ratio of the output voltage to the input voltage with all channels connected in parallel and off (usually expressed as an input-to-output attenuation ratio in decibels).

To realize a large number of multiplexed channels, you can connect analog multiplexers in parallel (called single-level multiplexing) using the enable input to control each device. It is also possible to connect the output of several multiplexers to theinputsof another (called double-level multiplexing) to expand the number of channels.

DIGITAL-TO-ANALOG CONVERTERS

D/A converters (DACs) are the devices used by computers for communications with the **real** world. They are employed in a variety of ways, such as for CRT displays, voice synthesis, automatic test systems, digitally controlled actuators, and so on. To understand how such a device works, consider the transfer function for a 3-bit DAC shown in Figure 9.



gure 11 --Most A/D and D/A ***converters troduce some degree ofoffset, gain, and/ of nonlinearity error.***

Here, each input code word produces a single discrete analog output value, generally as a voltage. There are 2^n different values produced including zero, with that output having a one-to-one correspondence with the input.

There are many different approaches used for DAC implementation, but most in use are of the parallel type, where all bits change simultaneously upon application of an input code word. Serial-type converters, on the other hand, produce an analog output only after receiving all digital input data in sequential form.

To employ a DAC within our process control application (see Figure 1), I have chosen to describe a configuration that provides a deglitched analog signal to our feedback control device actuator (of the pressure valve or heating element). All D/A converters produce output spikes, or glitches. They are most serious at transitions of 1/4, 1/2, and 3/4 of the full scale (*fs*). An example of this is shown in Figure 10. They are caused by the small time differences between current **sources** being turned off and on within the D/A converter. Take for example the major code transition at half scale from 0111.. .1111 to 1000...0000. Here the MSB (leftmost bit in the converter) current source tumson while all othercurrent sources are off. As a result, the small difference in switching time causes a narrow half-scale spike which could cause a hiccup in a finely tuned actuator.

To overcome this problem, the digital input to the DAC is controlled by an input register while the converter output is stored on a sample-and-hold. When the digital input is upgraded by the register, the sample-and-hold is switched into the hold mode. After the DAC has changed to its new value with all spikes settled out, the sample-and-hold is then switched back into a tracking mode. As a result of this process, the output changes smoothly from its previous value to the new value.

ERROR FUNCTIONS IN A/D AND D/A CONVERTERS

Of course, actual A/D and D/A converters don't have the wonderful/ ideal transfer functions I've described above. In reality they contain three different types of operational errors, called offset, gain, and nonlinearity errors. They are all functions of time and temperature and they all appear simultaneously.

Examples of these error functions are shown in Figure 11. They are defined as:

offset error results from transfer function failure to pass through zero.

gain error is the difference in slope between the actual transfer function anditsidealone.Itisusuallyexpressed as a percent of analog magnitude.

nonlinearity **error** is defined as the maximum deviation of the actual transferfunctionfromanidealstraight line at any point along the function. It is expressed as a percent of the full scale or in least-significant bit size. It assumes that the offset and gainerrors have been adjusted to zero.

Today, most A/D and D/A converters enable external trimming of offset and gain errors, at least at ambient temperature. Nonlinearity error, however, cannot be adjusted out and remains an inherent characteristic of the converter.

But remember, be careful when you operate a converter over a significant temperature range. Theeffect due to this change must be carefully determined. Of key importance is whether the device remains monotonic; that is, whether it has missing codes. This can be determined by computing the "differential nonlinearity tempco" specified for the converter. If you assume that the converter has at least half the least-significant bit of differential nonlinearity error, then the change in temperature resulting in an increase to 1 can be written as,

$$\Delta t = \frac{2^{-n} \times 10^6}{2DLT}$$

where n is the converter resolution in bits and *DLT* is the specified differential nonlinearity tempco in parts per million (ppm) of full-scale range per degree Celsius. Here, t is the maximum change in temperature that the convertercanhandlebeforeitbecomes non-monotonic.

This suggests that an organized approach be undertaken when you design your DAAC system. The A/D and D/A converters are the primary components. so you should draw up a checklist of required characteristics which, at minimum, should include the following key items:

1. converter type
2. resolution
3. speed
4. temperature coefficient

After your choice has been narrowed down by these considerations, you should then determine how your system fits the following:

1. analog signal range
2. type of coding
3. input impedance
4. power supply requirements
5. digital interface required
6. nonlinearity error
7. output current drive
8. type of start, stop, and status signals
9. size and weight

APPLICATIONS OF DAAC SYSTEMS

Recently there has been marked enthusiasm for using microcomputers for data acquisition and control. It is currently **estimated** that **nearly 40% of** all computer applications require some form of analog I/O. And of that, it's estimated that more than half use a PC in one form or another. Of course, the actual application determines both the type and number of analog signals involved. In most industrial process control or monitoring operations, hundreds of control points are continuously processed. These systems usually don't lend themselves to a PC environment because of the large number of data channels and the need for fast multitasking environments. However, in simpler systems, where the number of data channels is less **than 100, the PC has come into its own.** This is where the steadily increasing computing power of the PC has made consistent inroads. Systems with fewer analog channels allow having the analog circuity located directly on the same printed circuit board as the I/O interface logic. The card is made to just drop into any of your 16-bit or 32-bit slots with the analog input coming directly from the amplifier or your transducer. A multiplexer in front of the card enables the PC to handle multiple channels on the same card. The result is that this unit can be treated as a standard I/O peripheral using canned software drivers. Examples of different peripheral configurations that you can find on the market are:

- Analog-in (A/D only) with 8- to 256-channel input capability.
- Expanders for applications needing extra analog channels.
- Analog-out (D/A only) with 4- to 64-channel output capability.
- Analog-out expander boards for multichannel analog out capability.
- Combination A/D/A units with 8 to 64 analog I/O channels.

To be honest, there are other configurations, but you get the point. There is a whole realm of real PC-oriented application DAAC system components available. The availability of hardware and associated driver software has made the "desktop" PC a necessary part of the well-equipped lab or shop floor. ❖

Chris Ciarcia has a Ph.D. in experimental nuclear physics and is currently working as a staffphysicist at a national lab. He has extensive experience in computer modeling of experimental systems, image processing, and artificial intelligence. Chris is also a principal in Tardis Systems.

IRS

403 Very Useful
404 Moderately Useful
405 Not Useful

FEATURE ARTICLE

Tim **McDonough**
& Bruce Webb

An Interface for Portable Battery-Backed RAM

Using the Mitsubishi BEE Card for Nonvolatile Data Storage

"You can't take it with you."

Not so long ago, the adage about wealth and death held with equal validity for portable or remote data gathering applications. Portable units lacked the power to support floppy disk drives, and networks are simply not feasible in every situation. Fortunately, engineers have provided several possible solutions to the data portability problem in recent months. One solution gaining acceptance is the "data card"-a credit card-sized device containing battery-backed RAM.

Credit card-style RAM cards have recently appeared in such consumer products as video games and even personal computers such as the Atari Portfolio pocket computer. These applications typically use a mask-programmed ROM card or a similar one-time-programmable ROM card, but the RAM cards are also available in EEPROM and battery-backed SRAM versions that make them ideal for storing and transporting information. Currently available RAM card storage capacities range from 8K bytes to 1 megabyte of storage.

Recently, we designed a basic general-purpose single-board computer, the Datalog-R I (pronounced "data logger, one"; see Photo 1), that could be used for simple process-control functions and would also provide nonvolatile, removable storage. We chose to incorporate a 32K-byte "BEE card" from Mitsubishi Plastics in our design.

The SRAM version of the BEE card we used has 32K of 250-ns CMOS static RAM (Mitsubishi part no. 0256SRMSP25). Power consumption is 200 milliwatts maximum at 5 volts and the package has a rated operating temperature of 0–60°C. Each card measures 54 mm wide by 86 mm long and is approximately 2.3 mm thick. The card includes a holder for a thin, circular lithium battery (type CR-2016) that provides power to retain the cards data any time external power is not supplied. Mitsubishi rates battery life at 4 years when operated at 25°C. A write-protect switch, located on the end of the card, can be used to protect data from being accidentally overwritten if desired.

A 32K BEE card was chosen for several reasons. First, using the BEE card as a mass storage device is very straightforward. It can be read from and written to just like any other RAM device in the computer's memory map. There are no special disk drive interfaces to build and no disk BIOS to write as there would be had we incorporated a disk drive. Selecting the 32K size allowed us to place some conventional EPROM and SRAM on the board and still address the BEE card within the 8031's 64K address space without adding any sort of a bank switching scheme.

The second reason for using the BEE card is it has no moving parts such as drive motors that generate heat, collect dirt, and make noise, or heads to go out of alignment. Since the BEE card is all solid-state, battery-backed SRAM, it has none of these shortcomings. Most of the flexibility of toting around a floppy disk is retained, although in our case not quite as much data storage is available.

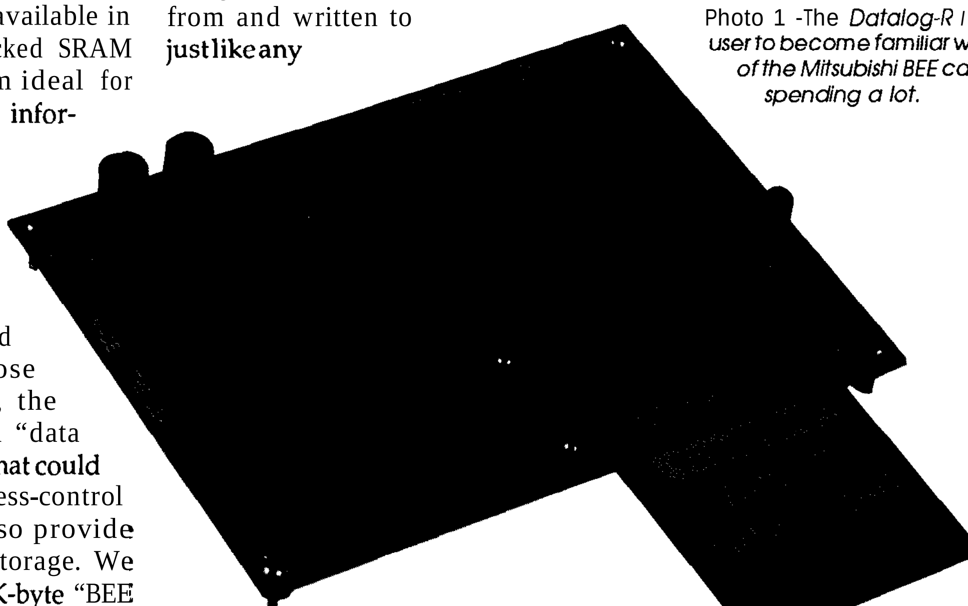


Photo 1 -The *Datalog-R I allows the user to become familiar with the use of the Mitsubishi BEE card without spending a lot.*

8031 TO THE RESCUE

A schematic of our design is shown in Figure 1. The size of the finished board, without the BEE card inserted, is 4.5" by 6.0". A 34-pin header provides access to ports 1 and 3 of the 8031 as well as providing access to the 5-volt DC power bus and ground connections. Additional connections are provided for an RS-232–compatible serial communications port and an external reset switch for the 8031.

Aside from the use of the BEE card, the system is a fairly conventional 8031-based single-board computer that should be quite familiar to CIRCUIT CELLAR INK readers. A 74LS138 address decoder provides selection of 8K memory devices at 0000H, 2000H, 4000H, and 6000H. Jumper blocks on the *CE line of each 28-pin socket let the user select the base address of each package depending on individual needs.

Two gates of a 74LS00 NAND package are used in conjunction with the 8031's *PSEN and *RD signals to derive a three-wire bus that can be used to configure on-board memory in one of three ways. The jumpers allow each 8K device to be designated as RAM or EPROM like you might expect. A third position, "BOTH," causes the 8031's Program and Data memory spaces to be overlapped within a given 8K block. This option is provided to allow compatibility with certain versions of Forth and other high-level languages that require a single, combined memory space for proper operation.

A jumper is also provided on pin 31 of the CPU to enable or disable the internal ROM found in some 8051 family microcontrollers. This makes it possible to use mask-programmed 8031 microcontroller derivatives such as the Micromint 80C52-BASIC CPU or the Intel 8052AH-BASIC CPU.

A third gate in the 74LS00 package is used to generate a *BCS (BEE Card Select) signal from address line 15 of the CPU. Since the BEE card serves as a mass storage device in this system, the RAM/ROM/BOTH options are not jumper selectable.

Other connections to the BEE card are similar to any other memory device. *WR, *RD, high-order address lines, and demultiplexed data and low-order address lines are all run to the special 32-contact socket where the BEE card itself is inserted.

The printed circuit board for the Datalog-R I includes space for a 5-volt DC regulator to make it simple to run the data logging system from an external 12-volt power source. Twelve volts DC is readily available in vehicles and is easily obtained in out-of-the-way locations where instrumentation and sensing equipment are often powered by photovoltaic arrays that charge secondary storage batteries.

MOVING THE DATA

Although perhaps not typical of a logging device, we chose to equip our design with an RS-232–compatible

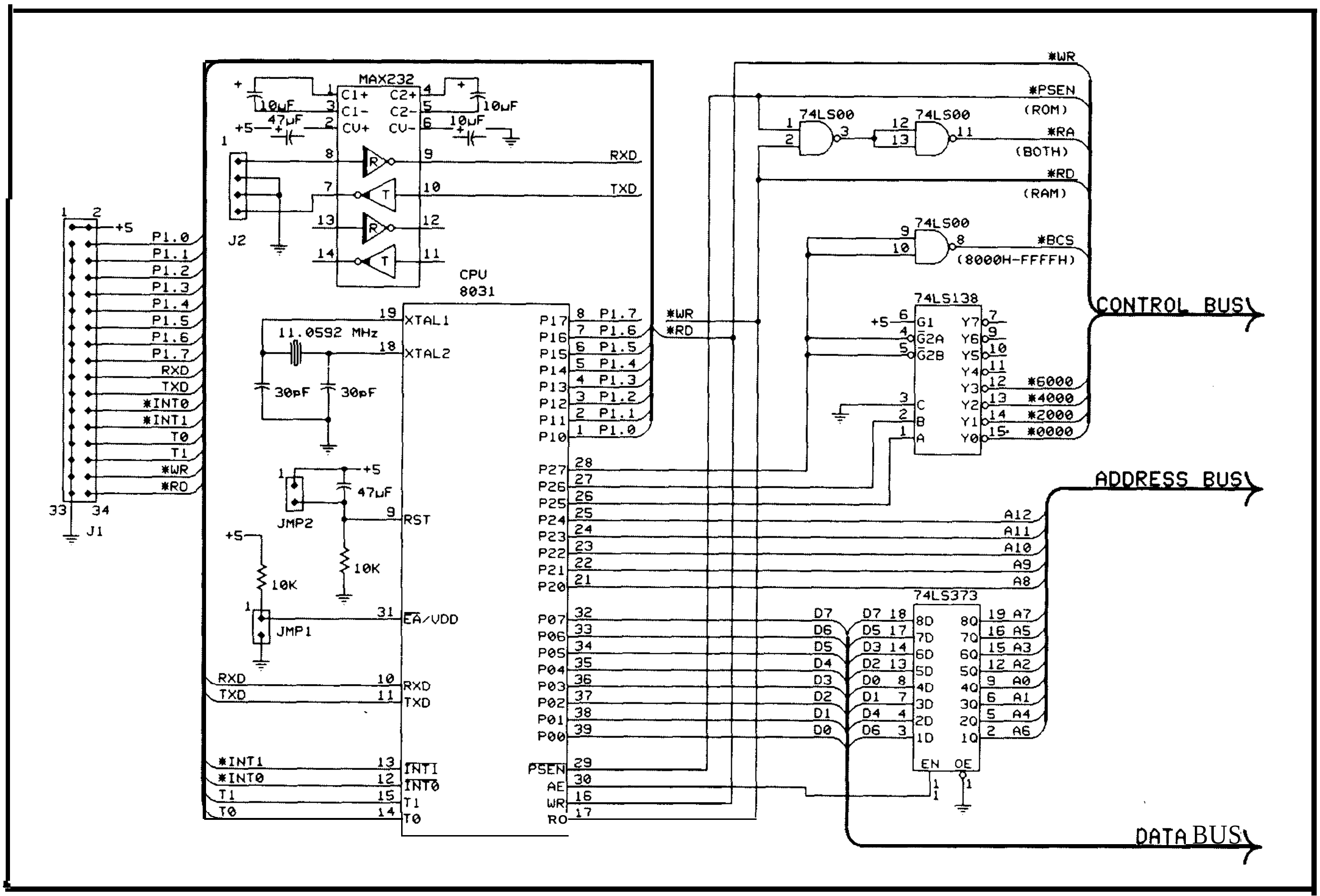


Figure 1a—*The Datalog-R I consists of a basic 8031 circuit with a simple expansion header.*

communications port. The port is implemented using the 8031's built-in UART and a Maxim MAX232 to maintain single-voltage power supply operation. An 11.0592-MHz crystal allows no-hassle programming of standard data rates from 1200 through 9600 bps.

A serial port makes sense in this application for a couple of reasons. First, and most important, it allows a Datalog-R board with appropriate firmware to be used as an inexpensive BEE card reader back at the lab so the user can move the collected data to another computer for analysis and further processing. Second, we recognized that there is a lot of scientific instrumentation equipped with communications ports that the original manufacturer assumed would be connected directly to a host computer. By using a Datalog-R as a sort of "mini host" that polls the equipment for information and stores it in the BEE card, it is possible to use the equipment in a remote location without having to dedicate a PC or an expensive laptop computer to data collection when the total volume of data is small.

IN THE FIELD

Using the Datalog-R is quite simple. Although we have inserted and removed the RAM card from an operating system many times, it is advisable to have the power off whenever swapping cards. This avoids the possibility of inserting or removing the card while it is being accessed by the 8031 and perhaps corrupting the data or damaging the card.

The BEE card is inserted in the Datalog-R with the legends and the electrical contacts facing up. Once a card is inserted, power is applied to the computer and the system is reset. In most cases customers have developed their final applications to begin running immediately after reset, requiring no further operator intervention.

A 32K BEE card holds a lot of data if you choose your storage algorithm correctly. In a typical temperature loggingapplication, a thermistor-type probe might be sampled four times per hour and its 8-bit value stored along with the time and date to the BEE card. Even without doing any clever programming to compress the information, data stored in a format such as YYMMDDHHMM<data byte> would allow for an entire month's (31 days) worth of data to be collected before the BEE card needed to be changed. Throw out the year and the month, or write them only once as "header" information whenever the system is powered up and the storage time can easily be stretched to fourty-eight days!

Our approach with this design was to build a system that was appropriate for smaller, simple applications. It has helped us solve some unique problems and we've gained valuable experience regarding the use of "RAM card" technology along the way.❖

We would like to thank Claire Bienen at Mitsubishi International for helping out with technical information, support, and of course the BEE cards used in our original prototype. Many thanksalsogo to Carl Baxter of Business and Technical Consultants for his real-world testing of several early prototypes.

Tim McDonough and Bruce Webb are principals in Cottage Resources Corporation.

Sources

Mitsubishi International, Inc.
520 Madison Avenue
New York, NY 10022

An assembled and tested Datalog-R I (less the 32K BEE card) is available for $179.95 (plus $3.00 S&H per order). 32K BEE cards are available for $57.95 each when purchased with the Datalog-R I.

Cottage Resources Corporation
1405 Stevenson Dr., Ste 3-672
Springfield, IL 62703
(217) 529-7679

IRS

406 Very Useful
407 Moderately Useful
408 Not Useful

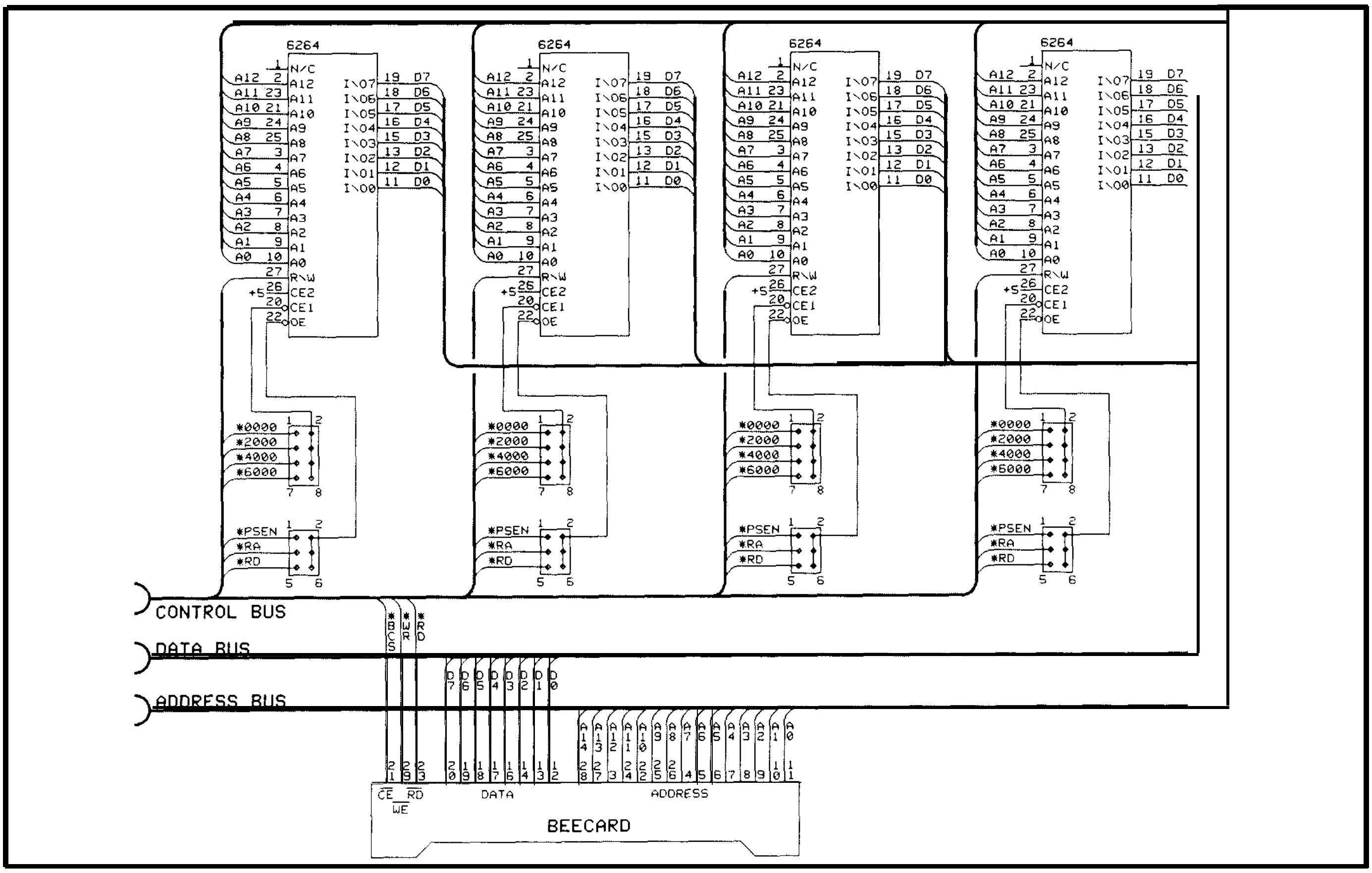


Figure 1 **b—*The second half of the schematic shows a bank of on-board RAM and EPROM plus the socket for the BEE card.***

FEATURE ARTICLE

Steve Ciarcia & Burt Brown

Using the Motorola MC68HC11

A Venerable History and a Certain Future

I come from the school that says all computers are equal, some just run a little faster than others. Provided that the price is not the overriding issue (such as 8088 chips suddenly being regularly priced at 39¢ each), I generally choose a processor for my projects based on ease of use and ease of explanation. In point of fact, however, most of my projects treat the microcontroller as a generic device. I may state the processor type but I use high-level languages so that the code is transportable and not processor specific.

For me, development tools and low-cost high-level language support take precedence over issues like brand loyalty. As most of you already know, I don't like to program any more than I have to: I usually use a controller board with an 80C52-BASIC processor that can be directly programmed in BASIC. Many of my applications-oriented projects have been greatly expedited by incorporating off-the-shelf BCC52 and RTC52 controllers rather than reinventing the wheel each time.

While I'll be the first to say that a computer is a computer, I temper that assertion with a little reality. When it comes down to fitting a variety of specific control functions on a few square inches of PC board, integration density and on-chip attributes make microcontroller chips very different. While the RTC52 generally meets all my programming objectives and can be expanded to include lots of I/O, multiple boards cannot always be accommodated in every application. I still see the need for a more highly integrated very low power single-board microcontroller for future projects.

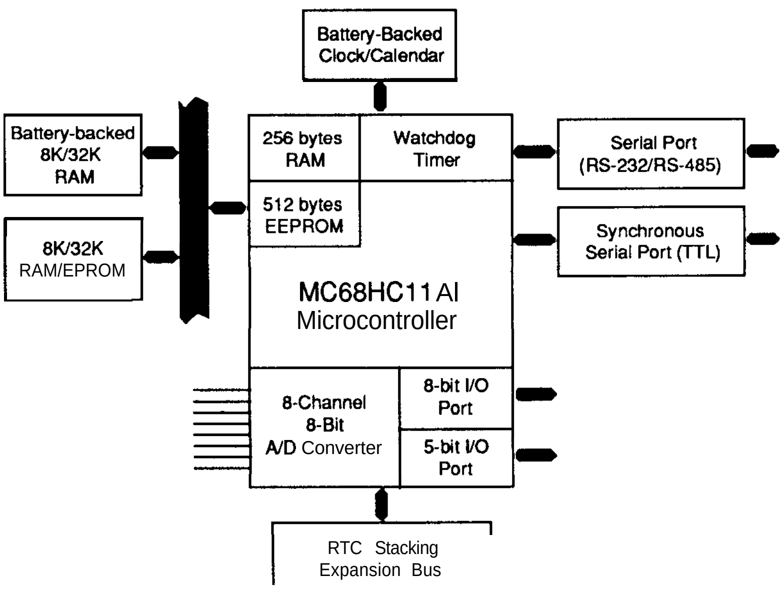


Figure 1 —*Functional block diagram of the RTC-HC 11.*

At first I considered using the new Signetics 8031-based super chips and porting BASIC-52 to them, but I was advised that it would be an ordeal and that there were many incompatibilities. Instead, thanks to a vocal group of Circuit Cellar BBSers who offered opinions, I ended up selecting the Motorola 68HCll. The 68HCllAl chip (see sidebar) is a wonderfully functional microcontroller which offers considerable on-chip attributes.

Using the 68HC11A1, we were able to put functions like parallel I/O, battery-backed RAM, EPROM, EEPROM, watchdog timer, battery-backed clock/calendar, serial ports, and an A/D converter in a low-cost 3.5"x 4.5" form factor. Combined with high-level languages, cross-assemblers, and a monitor, this RTC-HC11 board offers a formidable platform for many future Circuit Cellar projects.

Of course, we have to start some place, and I can't just drop an RTC-HC11 into some project without detailing the design of it for you beforehand. After you familiarize yourself with the 68HCll from the sidebar, I'll fill you in on the specifics of the RTC-HC11 board architecture, attributes, and memory map. Finally, with help from Burt, I'll address some software, development tools, and programming examples for the RTC-HC11.

THE BASICS

The RTC-HC11 is a single-board, 8-bit computer/controller based on the Motorola MC68HC11A1 MCU. Like the 8031- and V25-based designs previously presented in the pages of CIRCUIT CELLAR INK, *[Editor's Note: See "Creating a Network-Based Embedded Controller," CIRCUIT CELLAR INK #8; and "PC Programming Comes to Embed-*

ded Control," Circuit Cellar INK #17.] the RTC-HC11 uses the same dual vertical-stacking I/O expansion bus. I took advantage of the 68HC11A1 to build the following features onto the board:

*Motorola MC68HC11A1 MCU running at 8.00 MHz
*Up to 21 bits of TTL-level I/O (8 bits are shared with ADC)
•Asynchronous serial port; either full-duplex RS-232 or half-duplex RS-485
•Synchronous serial port with data transfer rate of up to 1 MHz
• 8-channel, 8-bit A/D converter
•512 bytes of EEPROM
*Battery-backed real-time clock/calendar
•Up to 64K of on-board RAM/EPROM, 32K bytes of battery-backed RAM
*Five-volt-only operation
•Small 3.5" x 4.5" form factor
*Compatible with RTC-series I/O expansion boards

RTC-HC 11 HARDWARE

An RTC-HC1 1 computer/controller board is composed of the three distinct hardware subsystems shown in Figure 1. These include the MCU itself and its associated I/O and reset circuitry; external RAM, EPROM, and memory address decoding; and the on-board real-time clock/calendar with battery backup and write-protect circuitry.

At the heart of the RTC-HC11 board lies a PLCC-packaged Motorola MC68HC11A1 HCMOS microcontroller unit (MCU). This is a high-speed, low-power, fully static MCU design featuring over half a dozen sophisticated on-chip peripherals. In fact, all of the on-board I/O devices with the exception of the real-time clock arelocated within the MCU, and an RTC-HC11 board can be stripped down to just two chips (the 68HC11A1 and the MAX232) and still be able to run small assembly language programs.

All of the MCU's subsystems can be accessed by a program executing from anywhere in the processor's

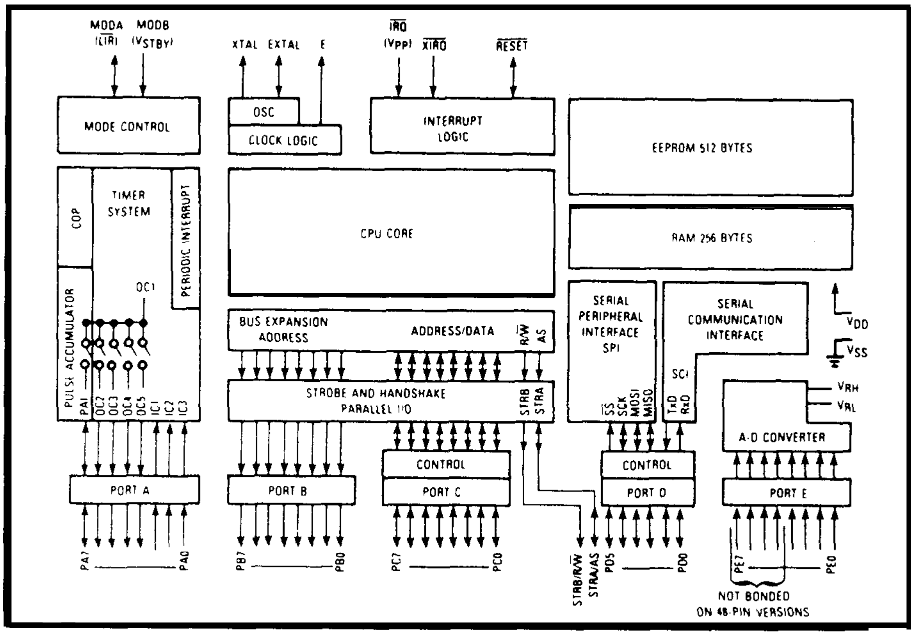


Figure 2—*The* block *diagram for the MC68HC11MCU shows a host of on-board devices.*

address space and while the MCU is in any of four different operating modes. (See the sidebar and Figure 2 on MCU operating modes for a more detailed discussion.) Included on the processor itself are a 16-bit timer subsystem with programmable prescaler and S-bit pulse accumulator; a standard NRZ asynchronous serial interface; a high-speed synchronous serial interface; an eight-channel 8-bit A/D converter; 512 bytes of EEPROM; 256 bytes of static RAM; and "Computer Operating Properly" (COP) and clock monitor watchdog timers.

In addition, there are seventeen **hardware** interrupts and one software interrupt with all on-chip interrupt sources being maskable. Two external inpu ts, IRQ (*INT1) and XIRQ (*INTO) are provided for interfacing to other devices with the IRQ input being programmable for either edge- or level-sensitive operation.

Software features include low-power STOP and WAIT modes which can be exited by internal or external interrupts and a special "wake-up" feature for using the asynchronous serial interface in networked systems. Double accumulators and index registers along with a full complement of bit manipulation and branch instructions are provided. Six different addressing modes allow for code optimization in all situations.

INTERNAL AND EXTERNAL MEMORY

Up to 64K of external memory can be used on the RTC-HC11 board. There are two sockets (U8 and U9), which can each accept an 8K or 32K static RAM or EPROM. A Dallas Semiconductor DS1210 power fail/battery backup chip converts one of the external memory sockets into nonvolatile memory for data logging or program development purposes. In a typical RTC-HC11 system (shown schematically in Figure 3), U8 will contain EPROM and U9 will contain RAM.

The 68HC11 MCU contains 256 bytes of internal RAM located at addresses $0000 through $0100. This memory region is also referred to as "page zero" RAM. Typically, a program will place data which needs to be frequently accessed in this area. There are also several machine language instructions which operate exclusively on data in this area. The Al-version MCU also contains 512 bytes of nonvolatile EEPROM memory. Each byte may be individually accessed and the entire memory array can easily be bulk erased. By default, the EEPROM is located at addresses $B600 through $B7FF. This can be changed by reprogramming a configuration register.

An 8K or 32K static memory chip may be inserted into U9 to provide

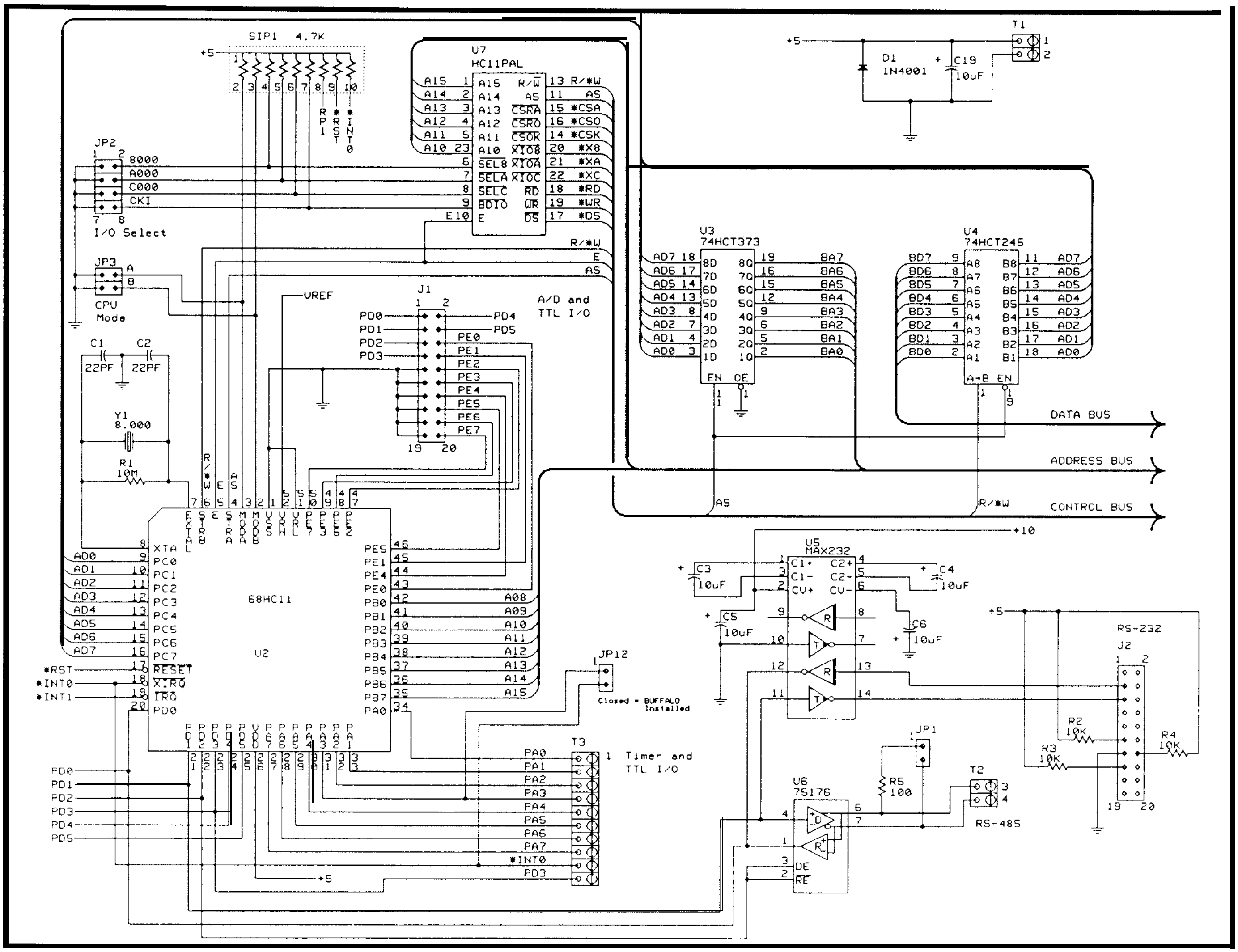


Figure 3a—*The* first *half of the RTC-HC11 schematic shows the processor, bus buffers, I/O decoding, and serial interfaces.*

programs with additional memory. The U9 socket is decoded in the range of $0000 to $7FFF. Reads or writes to this area are disabled by the MCU during any access to an on-chip storage location. Access is also disabled for addresses $7C00 to $7FFF by the U7 PAL (see Figure 4) if the I/O select jumper for the on-board real-time clock is installed. The ROM monitor (BUFFALO 3.4) does not require the use of external RAM. Several jumpers must be set to indicate the size of the installed chip and to enable the battery backup and write-protect circuitry.

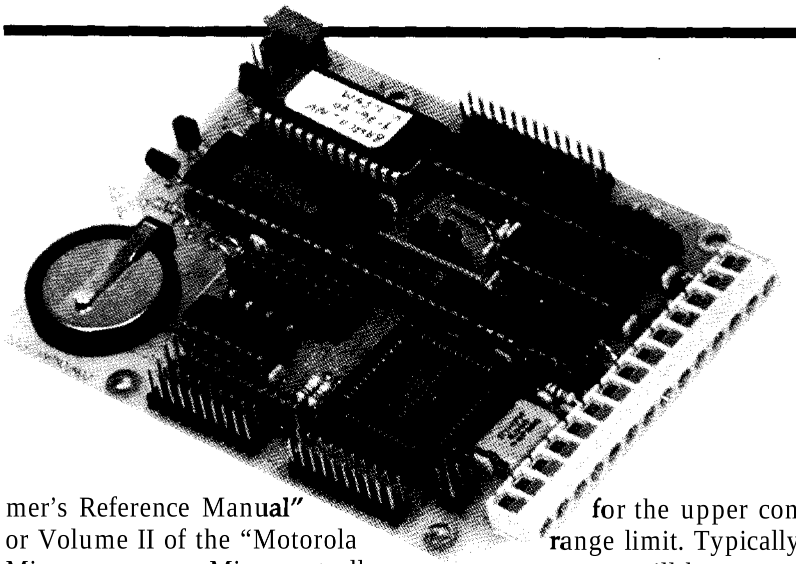
The U8 socket is decoded at addresses $8000 to $FFFF. During normal RTC-HC11 operating modes, the chip inserted into this socket will contain executable code and the MCU will fetch its reset vector from locations $FFFE and $FFFF. In some situations, it may be desirable to install a low-power static RAM in U8 and configure the battery backup jumpers to continuously maintain its contents.

GETTING IN (AND OUT)

Since the 68HC11 processor contains several on-chip peripheral subsystems, most of the external I/O lines are shared with at least one other subsystem. For example, any or all of the eight A/D converter inputs can also be used as TTL inputs. Thus, an application which required only four channels of A/D could use the remaining four channels as TTL input. This flexibility allows for allocation of on-chip resources with a minimum amount of waste. As a result, configuration of each on-chip subsystem becomes somewhat complicated with the 68HC11 having over sixty configuration registers, the details of which are beyond the scope of this article. (To fully utilize all the features of the 68HC11, I recommend obtaining a copy of either the "68HC11 Program-

```
OKI     = /A15 * A14   * A13   * A12   * A11  * A10 . BDIO
CSRAM   = /A15 . /OKI
X8000   = A15 * /A14   . /A13  . /A12  . SEL8 . E
XA000   = A15 * /A14   * A13   . /A12  * SELA * E
XC000   = A15 . A14    . /A13  . /A12  * SELC . E
CSROM   = A15 * /X8000 . /XA000 . /XC000
DS      = E
RD      = RW * E
WR      = /RW . E
```

Figure 4—*Use of a PAL greatly reduces the number of discrete gates on the board.*



mer's Reference Manual" or Volume II of the "Motorola Microprocessor, Microcontroller, and Peripheral Databook.")

The RTC-HC11 processor board can decode up to three 8K blocks at addresses $8000, $AOOO, and $C000 for use by off-board I/O devices or RTC-series expansion boards. In addition, a 1K I/O block which is fixed at address $7C00 can be enabled to allow access to the on-board real-time clock/calendar. If one or more external I/O blocks are enabled, the chip select lines to the corresponding on-board RAM or EPROM are held high whenever the external device is being accessed.

Name	Bit	Function	Connector(s)
PORTA	O-2	Input	T3-1—T3-8
	3-6	output	
	7	Input/Output	
PORTD	O-5	Input/Output	J1-1,3,5,7,2 (J1-4,T3-10)
PORTE	O-7	Input	J2-6,8,10,12,14,16
IRQ		Input	JP15-13
XIRQ		Input	T3-9, JP15-11
RESET		Input	JP6-1

Table 1—TTL-compatible ports on the MC68HC11A1.

Up to 21 bits of TTL (logic level) I/O can be utilized on the RTC-HC11 processor, however several of these lines are input-only, output-only, or shared with another subsystem. Table 1 is a brief description of each I/O port or signal which can accept a TTL-level input or output, as well as the connector(s) on which the signal appears.

The on-chip A/D converter can be used to measure analog inputs in the range of O-5 VDC. All channels are single ended (referenced to ground) with an adjustable precision reference for the upper conversion range limit. Typically, this reference will be set to 5.00 volts yielding a converter resolution of 5.00/256 or 19.6 millivolts per step for an 8-bit ADC. Source impedance for each channel should not exceed ten thousand ohms and total sample acquisition and conversion time for each channel is on the order of twenty-two microseconds. The ADC reference voltage is set by an LM336 reference diode set at 5.00 V. The LM336 is powered from the positive voltage output of the MAX232 RS-232 driver. This chip must be installed for the ADC to operate to its maximum range.

Each of the eight input channels can be read and the resulting voltage calculated as follows:

$$\text{Input voltage} = (V_{ref}/256) \times (\text{Port input value})$$

For example, if V_{ref} = 5.00 volts, and channel 0 of the A/D converter is read and its value is 200 (decimal), then the input voltage to channel 0 is 5.00/256 x 200 = 3.91 volts. Listing 1 illustrates the use of the analog-to-digital converter while in a BASIC-11 program.

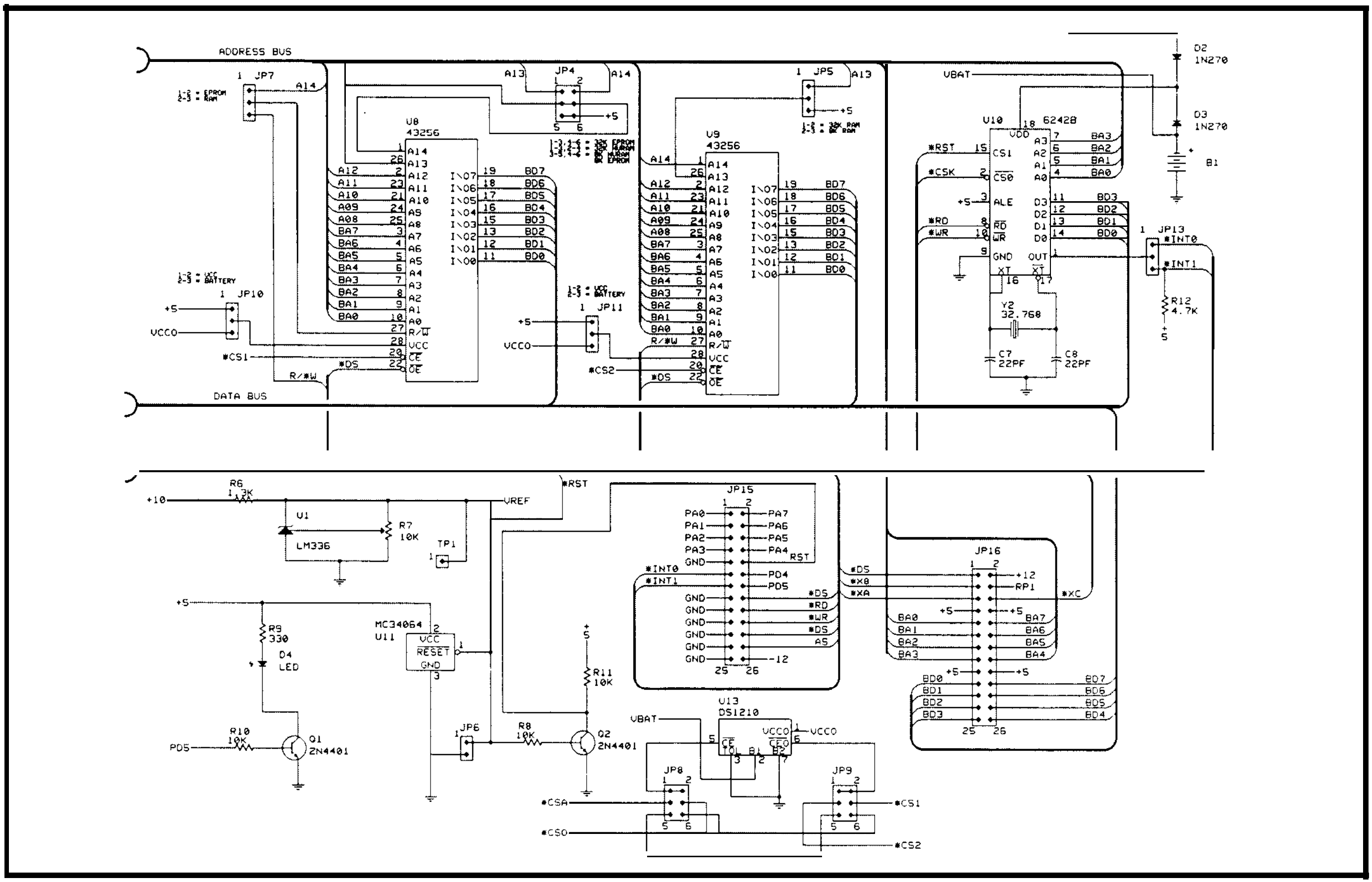


Figure 3b—*The second half of the RTC-HC11 schematic includes the external memory, real-time clock/calendar, and power supervisor circuitry.*

SERIAL OPTIONS

A standard asynchronous serial port with selectable baud rate, parity, character length, and stop bits is provided as an on-chip peripheral subsystem. This subsystem is externally interfaced through both a MAX232 RS-232 driver (U5) with true positive and negative voltage levels, and an SN75176 RS-485 driver (U6) capable of driving up to 6000 meters of 24 AWG, 2-conductor cable. Both interface chips are always enabled with Port D bit 2 controlling the direction of data flow on the RS485 interface. A logic one (1) on Port D, bit 2 places the RS485 interface driver into transmit mode. Another jumper terminates the RS-485 transceiver with a 100-ohm resistor.

Access to the serial port through BASIC-11 is accomplished simply by using BASIC's PRINT and INPUT statements since this is the default console port for the BASIC interpreter.

A four-wire (In, Out, Clock, Select), full-duplex high-speed synchro-

MC68HC11A1 SIGNAL DESCRIPTION

RESET-This active-low bidirectional control pin is used as an input to initialize the MCU to a known startup state and as an open-drain output to indicate that an internal failure has been detected in either the clock monitor or the computer operating properly circuit.

XTAL, EXTAL—Provide the interface for either a crystal or a CMOS-compatible clock to control the internal clock generator circuitry. The frequency applied is four times higher than the desired E clock rate.

E-Provides an output for the internally generated E clock, which can be used for timing reference. The frequency of the E output is one-fourth that of the input frequency at the XTAL and EXTAL pins.

IRQ-The asynchronous interrupt input to the MCU. Either negative edge-sensitive or level-sensitive triggering is program selectable. This pin is configured to be level sensitive during reset. An external resistor connected to VDD is required on IRQ.

XIRQ—Provides the capability for asynchronously applying nonmaskable interrupts to the MCU after a power-on reset (POR). During reset, the X bit in the condition code register is set, and any interrupt is masked until enabled by software. This input is level sensitive and requires an external pull-up resistor to VDD.

MODA/LIR AND MODB/VSTBY—During reset, these pins are used to control the two basic operating modes and the two special operating modes. The LIR output can be used as an aid in debugging once reset is completed. The open-drain LIR pin goes to an active-low state during the first E-clock cycle of each instruction and remains low for the duration of that cycle.

VRL and VRH-Provide the reference voltage for the A/D converter.

R/W/STRB—Provides two different functions, depending on the operating mode. In single-chip mode, the pm provides STRB (output strobe) function; in the expanded-multiplexed mode, it provides R/W (read-write) function. The R/W is used to control the direction of transfers on the external data bus.

AS/STRA—Provides two different functions depending on the operating mode. In single-chip mode, the pin provides STRA (input strobe) function, and in the expanded-multiplexed mode, it provides AS (address strobe) function. The AS may be used to demultiplex the address and data signals at port C.

INPUT/OUTPUT LINES (PA9–PA7, PB0–PB7, PC0–PC7, PD0–PD5, PE0–PE7)—These I/O lines are arranged into four 8-bit ports (A, B, C, and E) and one 6-bit port (D). All ports serve more than one purpose depending on the operating mode. Port functions are controlled by the particular mode selected. In the single-chip *mode* and bootstrap mode, four ports are configured as parallel I/O data ports and port E can be used for general-purpose static inputs and/or analog-todigital converter channel inputs. In the expanded-multiplexed mode and test mode, ports B, C, AS, and R/W are configured as a memory expansion bus.

PORT A-In all operating modes, port A may be configured for three input capture functions, four output compare functions, and pulse accumulator input (PAI) or a fifth output compare function. Each input capture pin provides for a transitional input, which is used to latch a timer value into the 16-bit input capture register. External devices provide the transitional inputs, and internal decoders determine which input transition edge is sensed. The output compare pins provide an output whenever a match is made between the value in the free-running counter (in the timer system) and a value loaded into the particular 16-bit output compare register. When port A bit 7 is configured as a PAI, the external input pulses are applied to the pulse accumulator system. The remaining Port A lines may be used as general-purpose input or output lines.

PORT B-In the single-chip mode, all port B pins are general-purpose output pins. Port B may also be used in a simple strobed output mode where the STRB pulses each time port B is written. In the expanded-multiplexed mode, all of the port B pins act as high-order (bits 8–15) address output pins.

PORT C-In the single-chip mode, port C pins are general-purpose input/output pins. Port C inputs can be latched by the STRA or may be used in full handshake modes of parallel I/O where the STRA input and STRB output acts as handshake control lines. In the expanded-multiplexed mode, Port C pins are configured as multiplexed address/data pins. During the address cycle, bits 0 through 7 of the address are output on PC0–PC7; during the data cycle, bits 0 through 7 (PC0–PC7) are bidirectional data pins controlled by the R/W signal.

PORT D-In all modes, port D bits O-5 may be used for general-purpose I/O or with the serial communications interface (SCI) and serial peripheral interface (SPI) subsystems. Bit

nous serial interface is available for communications with external I/O devices such as EEPROMs, ADCs, PLLs, or other RTC-HC11 processor boards. This interface can transfer serial data at up to 1 MHz; clock phase and polarity are software programmable. Because its operation is relatively complicated, you'll excuse me for glossing over its relevance and suggesting that you seek out a Motorola "68HC11 Programmer's Reference Manual" for more details.

YOU WANT IT WHEN?

Data collection and control applications are usually time dependent. The addition of a hardware clock/calendar relieves the microcontroller from timekeeping overhead. The Oki M6242B CMOS clock/calendar (U10) has both a clock/calendar and selectable interrupt outputs. Thirteen registers hold time and date information and three registers are used for control purposes. These registers are addressed by the latched addresses BA0–BA3. To enable the chip, I/O select jumper JP2 pins 7 and 8 must be connected. This decodes a 1K I/O block for access to the clock chip beginning at address $7C00. Table 2 is the complete clock/calendar register address/function table. Each register is a nibble (4 bits) wide with the lower 4 bits of the data values significant. Each value is between 0 and 15; most are from 0 to 9 (a decimal digit).

The STD.P output from the M6242B can be used as an interrupt source. The upper part of JP25 allows selection of either *INT0 or *INT1 for the interrupt. Two modes of interrupt can be selected through the control registers D-F of the M6242B. The IRT (interrupt) mode gives a one-time interrupt pulse while the ST (standard) mode creates a recurring interrupt pulse.

DON'T FORGET

The RTC-HC11 has been designed so that a Dallas Semiconductor DS1210 Power Monitor IC can be installed in socket U13. This chip monitors the 5-volt power supply and automatically

Reg	Name	Function	Addr
0	(S1)	Seconds	$7C00
1	(S10)	Tens of seconds	$7C01
2	(MI1)	Minutes	$7C02
3	(MI10)	Tens of minutes	$7C03
4	(H1)	Hours	$7C04
5	(H10)	Tens of hours	$7C05
6	(D1)	Days	$7C06
7	(D10)	Tens of days	$7C07
8	(MO1)	Months	$7C08
9	(MO10)	Tens of months	$7C09
10	(Y1)	Years	$7C0A
11	(Y10)	Tens of years	$7C0B
12	(W)	Day of the week	$7C0C
13	(CD)	control register D	$7C0D
14	(CE)	control register E	$7C0E
15	(CF)	control register F	$7C0F

able 2—*The functions and register addresses for the OKI M6242B CMOS clock/* calendar chip.

switches either U8 or U9 to battery power if the supply voltage falls below 4.75 volts. In addition, the R/*W line is held high to prevent data corruption when the changeover to battery power takes place. Jumpers JP8 through JP10 are used to configure either U8 or U9 for battery backup. (Note: You cannot back up both U8 and U9 at the same time. Also, if you elect to install thisoption, a low-power static RAM with a standby current draw of around 2-S µA must be used to prevent excessive current drain from the on-board battery. Higher current RAM chips will still work, but you will be replacing batteries far more often.)

RTC-HC11 SOFTWARE DEVELOPMENT

The MC68HC11 series is one of Motorola's most popular 8-bit microcontroller families and there areabundant software development tools. One telephone call to Motorola's Freeware bulletin board system provided us with an IBM PC-based cross-assembler, a ROM monitor, 68HC11 simulator, a very fast integer BASIC, and a C compiler. *[Editor's Note: The Motorola Freeware BBS can be accessed at (512) 891-3733, parameters N81. In addition, software for this article is available from* the Circuit *Cellar* BBS or on *Software On Disk #18: See page 92 for downloading and ordering information.] The* BASIC-11 interpreter required a few changes to its console I/O routines since it was

0 is the receive data input, and bit 1 is the transmit data output for the SCI. Bits 2 through 5 are used by the SPI subsystem.

PORT E-Used for general-purpose static inputs and/or analog-to-digital channel inputs in all operating modes.

RESETS-The MCU can be reset four ways: an active-low input to the RESET pin; a power-on reset function; a computer operating properly (COP) watchdog-timer timeout; and a clock monitor failure. The RESET input consists mainly of a Schmitt trigger that senses the RESET line logic level.

RESET PIN—To request an external reset, the RESET pin must be held low for eight Ecyc (two Ecyc if no distinction is needed between internal and external resets). To prevent the EEPROM contents from being corrupted during power transitions, the reset line should be held low while VDD is below its minimum operating level.

POWER-ON RESET (POR)—Occurs when a positive transition is detected on VDD. The processor remains in the reset condition until RESET goes high.

COMPUTER OPERATING PROPERLY (COP) RESET-The MCU contains a watchdog timer that automatically times out if not reset within a specific time by a program reset sequence. If the COP watchdog timer is allowed to timeout, a reset is generated, which drives the RESET pin low to reset the MCU and the external system.

The COP reset function can be enabled or disabled by setting the control bit in an EEPROM cell of the system **configuration** register. Once programmed, this control bit remains set (or cleared) even when no power is applied, and the COP function is enabled or disabled independent of resident software.

CLOCK MONITOR RESET-The MCU contains a clock monitor circuit which measures the E clock input frequency. If the E clock signal is lost or its frequency falls below 10 kHz, then an MCU reset is generated, and the RESET pin is driven low to reset the external system.

INTERRUPTS-There are seventeen hardware interrupts and one software interrupt (ex-cluding reset-type interrupts) that can be generated from all the possible sources. These interrupts can be divided into two categories: maskable and nonmaskable. Fifteen of the interrupts can be masked with the condition code register 1 bit. All the on-chip interrupts are individually maskable by local control bits. The software interrupt is nonmaskable. The external input to the XIRQ pin is considered a nonmaskable interrupt because, once enabled, it cannot be masked by software; however, it is masked during reset and upon receipt of an interrupt at the XIRQ pin.

ANALOG-TO-DIGITAL CONVERTER-The MCU contains an g-channel, multiplexed-input, successive approximation, analog-to-digital (A/D) converter with sample and hold. Two dedicated lines (VRL and VRH) are provided for the reference supply voltage input. These pins are used instead of the device power pins to increase the accuracy of the A/D conversion.

The g-bit A/D conversions of the MCU are accurate to within +1 LSB (+1/2 LSB quantizing errors and +1/2 LSB all other errors combined). Each conversion is accomplished in 32 MCU E-clock cycles. An internal control bit allows selection of an internal conversion clock oscillator that allows the A/D converter to be used with very low MCU clock rates. A typical conversion cycle requires 16 microseconds to complete at a 2-MHz bus frequency.

Four result registers are included to further enhance the A/D subsystem along with control logic to control conversion activity automatically. A single write instruction selects one of four conversion sequences, resulting in a conversion complete flag after the first four conversions. Simply convert one channel four times and stop; sequential results are placed in the result registers.

AUTOST	CALL()	PORTD	IF/THEN/ELSE
ADC()	TIME	CLEAR	LIST
PRINT	DIM	ABS()	RTIME
CONT	LLIST	FOR/NEXT	PEEK()
FDIV()	PACC	DATA	NEW
WHILE/ENDWH	POKE	RND()	EEP()
NOAUTO	RETURN	REM	SGN()
ELOAD	ONIRQ	RETI	TRON
CHR$()	ESAVE	ONTIME	STOP
TROFF	HEX()	FREE	PORTA
SLEEP	ONPACC	HEX2()	INBYTE
PORTB	GOT0	ON..GOTO	TAB0
INPUT	PORTC	GOSUB	ON..GOSUB

Table J-BASIC- *11 keyword summary.*

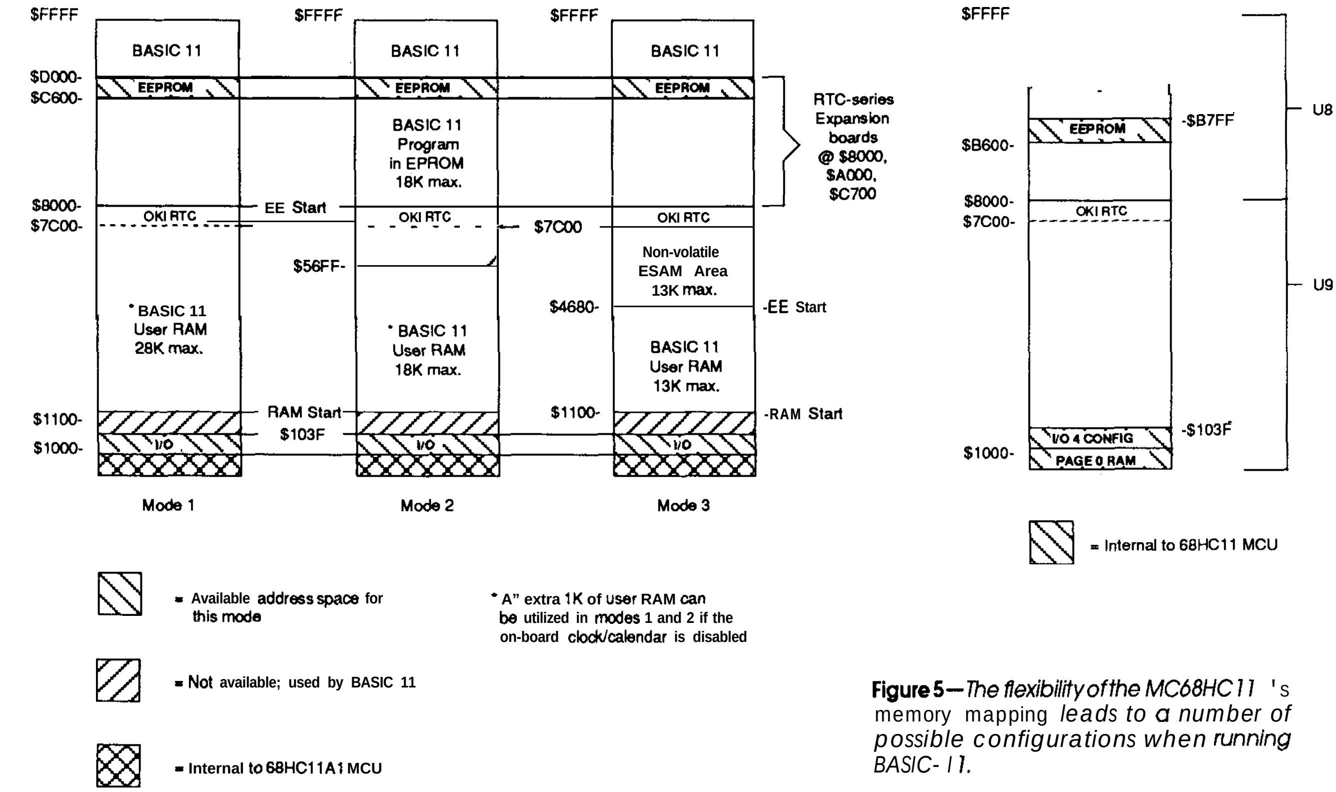


Figure 5—*The flexibility of the MC68HC11's memory mapping leads to a number of possible configurations when running BASIC-11.*

originally designed to run on Motorola's MC68HC11EVB, but it's a real performer on the RTC-HC11. Changes were also made to allow BASIC-l 1 to take advantage of the RTC-HC11's on-board battery-backed RAM and real-time clock/calendar chip.

The following sections contain a brief summary of each of the BASIC and assembly language development options and a few examples of how an embedded application can be configured for the RTC-HC11.

BASIC-l 1

For many of us, the quickest way to develop embedded software will be to use BASIC-11, a fast integer BASIC interpreter with built-in support for most of the 68HCll's on-chip peripherals. This BASIC can execute a thirty-thousand-iteration FOR...NEXT loop in less than four seconds (an IBM PC takes 38 seconds by comparison) and has built-in keywords for access to the I/O ports, interrupt request lines, timer/pulse accumulator, A/D converter, and EEPROM. Using battery-backed RAM, a BASIC-l 1 application can be directly saved and automatically executed on reset or power up, eliminating the need to repeatedly program and erase EPROMs. A BASIC-1 1 keyword summary appears in Table 3.

Listing 1 is a simple BASIC-11 example which uses the RTC-HC11's ADC, EEPROM, and timer subsystems. Line 40 sets up an interrupt service routine and uses the 68HC11's timer subsystem to generate an interrupt every hour. When the service routine at line 500 is called, the ADC is sampled and the readings are printed to the console (serial port) and also saved in EEPROM. Line 500 resets the ONTIME setup so that BASIC's TIME variable continues to increment properly.

Once a program has been debugged, several options are available for permanently saving the tokenized code depending on how the inter-

```
10 REM Stores all eight A/D converter channel readings to the
20 REM first eight EEPROM locations once each hour.
30 TIME=0
40 ONTIME 3600,500
50 GOSUB 700
60 GOTO 50
500 ONTIME TIME+3600,500
510 PRINT "ch 0","ch 1","ch 2","ch 3","ch 4","ch 5","ch 6","ch 7"
520 PRINT "---------------------------------------------------------"
530 FOR I=0 TO 7
540    A=ADC(I)        : REM read the A/D converter
550    PRINT A,        : REM send the reading to the console
560    EEP(I)=A        : REM save reading in nonvolatile EEPROM
570 NEXT I
580 PRINT : PRINT
590 RETI
700 REM This subroutine is called whenever the timer interrupt
710 REM is NOT being serviced (i.e., most of the time for a one-
720 REM hour timer)
730 RETURN
```

Listing 1—*Using BASIC-11's ADC and EEP functions allows BASIC to directly exercise the A/D and EEPROM.*

preter's configuration bytes have been set. Ultimately, my intention is to have configuration bytes that are located in the on-chip EEPROM which determine how RAM and EPROM memory will be shared by the BASIC-11 interpreter (initial software releases will involve separate EPROMs for each operating mode). In the simplest configuration (Mode 1), BASIC is given access to all available RAM from $1040 to $7FFF. This allows you to develop the largest possible RAM-based program, about 28K, and is the best choice if your application does not require nonvolatile program storage or automatic loading on power up. Volatile program storage is not a problem if you use a terminal emulator that supports ASCII transfers on a PC for communications. Simply list the program and save it to disk before powering down the RTC-HC11.

ASM	[<addr>]	Line asm/disasm
BF	<addr1> <addr2> [<data>]	Block fill memory
BR	[-][<addr>]	Set up bkpt table
BULK		Erase EEPROM
BULKALL		Erase EEPROM and CONFIG
CALL	[<addr>]	Call subroutine
GO	[<addr>]	Execute code at addr
PROCEED		Continue execution
EEMOD	[<addr> [<addr>]]	Modify EEPROM range
LOAD	[T]	Load S-record from terminal
VERIFY	<host dwnld command>	Load or verify S-records
MD	[<addr1> [<addr2>]]	Memory dump
MM	[<addr>] or [<addr>]	Memory modify
MOVE	<s1> <s2> [<d>]	Block move
OFFSET	[-]<arg>	Offset for download
RM		Register modify
STOPAT		Trace until addr
T	[<n>]	Trace n instructions

Table 4—*BUFFALO* 3.4 ROM *monitor command summary.*

Some people like to keep their program and data'on instant call. Storing the code in ROM or battery-backed RAM is the alternative. This is where Mode 2 (the "EPROM" mode) and Mode 3 (the battery-backed RAM mode) come into play.

Referring to the memory map in Figure 5, note that BASIC-11 itself uses up about 9K of the available ROM space leaving 23K in which to store a ROM-based BASIC program. Since the RTC-HC11 has only two memory sockets, careful attention must be given to the values used to configure BASIC-l l's `RAMstart` and `EEstart` to help eliminate the possibility of being able to create a program which is too large to store in either EPROM or battery-backed RAM. Second, some way to transfer the program out of the RTC-HC11's RAM and into an external EPROM is required.

One solution is to use the program shown in Listing 2. This code is designed to be appended to the end of your debugged application and then entered via a `GOTO` to line 31000. The program will dump a Motorola S-record format file to the serial port which

```
31000 REM This program will dump a combined user program and
31010 REM BASIC11 interpreter image at a start address of $8000.
31020 REM AUTOST will be set;  store $55 to $1040 for debug mode
31030 REM ***** gather info from page-zero memory *****
31040 REM
31050 SH=PEEK($0004)*256
31060 SL=PEEK($0005)
31070 BB=SH+SL
31080 IF PEEK($1040)=$55 THEN 31090 ELSE 31100
31090 PRINT "BASIC start address is: $";HEX(BB)
31100 SH=PEEK($0006)*256
31110 SL=PEEK($0007)
31120 BE=SH+SL
31130 IF PEEK($1040)=$55 THEN 31140 ELSE 31150
31140 PRINT "BASIC end address is: $";HEX(BE)
31150 SH=PEEK($0008)*256
31160 SL=PEEK($0009)
31170 VB=SH+SL
31180 IF PEEK($1040)=$55 THEN 31190 ELSE 31200
31190 PRINT "VAR start address is: $";HEX(VB)
31200 SH=PEEK($000A)*256
31210 SL=PEEK($000B)
31220 VE=SH+SL
31230 IF PEEK($1040)=$55 THEN 31240 ELSE 31250
31240 PRINT "VAR end address is: $";HEX(VE)
31250 SH=PEEK($000C)*256
31260 SL=PEEK($000D)
31270 HL=SH+SL
31280 IF PEEK($1040)=$55 THEN 31290 ELSE 31330
31290 PRINT "Highest line # is: ";HL;" ($";HEX(HL);")"
31300 REM
31310 REM ***** print first line of combined EPROM S-Record *****
31320 REM
31330 VC=VE
31340 CK=0
31350 PA=$8000 : REM start address of combined EPROM
31360 IF PEEK($1040)=$55 THEN 31370 ELSE 31390
31370 PRINT "Combined EPROM start: $";HEX(PA)
31380 PRINT
31390 RS=(PEEK($FFC0)*256)+PEEK($FFC1)  : REM get RAMstart constanl
31400 PRINT"S1";:PRINT"0E";:PRINT HEX(PA);:REM start S-rec print
31410 CK=CK+$0E+$80 : REM update checksum
31420 BB=BB-RS  : REM subtract RAMstart from BASBEG
31430 CK=CK+(((BB.AND.$FF00)/256).AND.$FF)+(BB.AND.$FF):REM chksum
31440 PRINT HEX(BB); : REM print new BASBEG for EPROM
31450 BE=BE-RS
31460 CK=CK+(((BE.AND.$FF00)/256).AND.$FF)+(BE.AND.$FF)
31470 PRINT HEX(BE); : REM print new BASEND
31480 VB=VB-RS
31490 CK=CK+(((VB.AND.$FF00)/256).AND.$FF)+(VB.AND.$FF)
31500 PRINT HEX(VB); : REM print new VARBEGIN
31510 VE=VE-RS
31520 CK=CK+(((VE.AND.$FF00)/256).AND.$FF)+(VE.AND.$FF)
31530 PRINT HEX(VE); : REM print new VAREND
31540 CK=CK+(((HL.AND.$FF00)/256).AND.$FF)+(HL.AND.$FF)+$55
31550 PRINT HEX(HL);"55"; : REM print new HILINE and AUTOST flag
31560 CK=CK.EOR.$FFFF
31570 PRINT HEX2(CK.AND.$FF): REM print checksum
31580 REM
31590 REM     ***** print S-records for user program *****
31600 REM
31610 SA=BB+RS
31620 HA=VE+RS
31630 OF=$6F0B : REM set to $6F0B for a user RAM start of $1100
31640 GOSUB 32000
31650 REM
31660 REM     ***** print S-records for BASIC interpreter *****
31670 REM
31680 SA=$D000
31690 HA=$FFFF
31700 OF=0
31710 GOSUB 32000
31720 PRINT "S9030000FC" : REM print termination record
31730 END : REM ***** EXIT *****
32000 REM
32110 CK=0 : REM S-record checksum
32120 REM
32130 IF PEEK($1040)=$55 THEN 32140 ELSE 32160 : REM debug flag
32140 PRINT:PRINT"Memory read from";"$";hex(SA);" to ";"$";hex(HA
32150 PRINT "S-Record starting address: ";"$";hex(SA+OF): PRINT
32160 LL=16 : REM # of data bytes in S-rec excl. address & cksum
32165 TB=HA-SA+1 : REM calculate total bytes
```

(continued)

Listing 2-S-record *dump program.*

can then be captured by a PC-based communications program like Procomm or Kermit. This S-record file will contain your BASIC application code from RAM, a copy of the interpreter itself from ROM, and a few bytes of information that BASIC needs to auto start the ROM-based application. This file can then be sent directly to an EPROM programmer to produce a 32K device (27256) which, when inserted into socket U8, will automatically run your BASIC-11 application on reset or power up.

The last configuration, Mode 3, allows a portion of battery-backed static RAM to be used for semipermanent program storage. Here's how it's done:

BASIC-l 1 has an ESAVE keyword which is intended to move a program from BASIC's program RAM to some category of nonvolatile external (to BASIC) RAM storage. ESAVE uses the value contained in an internal variable to determine where in memory to place the saved program. This internal variable can be changed by setting the fifth and sixth bytes of the on-chip EEPROM to a value other than $FF. These new EEPROM values represent the memory address at which BASIC-11 will attempt to store an ESAVEd program.

Since the RTC-HC11 has a Dallas Semiconductor DS1210 RAM backup-protection circuit, we can use some of BASIC's normal program RAM for the ESAVE storage area. This is accomplished by setting jumpers JP8–11 so that backup power is supplied to U9 from the on-board battery and the read/write line is routed through the DS1210's power-down protection circuitry.

This effectively creates an on-board EEPROM into which we can store/retrieve a BASIC-l 1 program by using the ESAVE and ELOAD commands. If we also issue the AUTOST command, the ESAVEd program will automatically execute on reset or power up. The stored program can be copied back to user RAM for editing by using the ELOAD command. As a bonus, the user RAM area can be used by the ESAVEd program for data storage.

ASSEMBLY LANGUAGE

The assembly language memory map (see Figure 5) is inspiring and uncluttered; kind of like pulling onto a five-lane freeway at three o'clock in the morning in a Porsche.

Each of the RTC-HC11's memory sockets can be decoded for an 8K or 32K memory block; U9 covers $0000–$7FFF, and U8 covers $8000–$FFFF. Three small memory blocks are decoded for peripheral devices: $1000–$103F for on-chip I/O and configuration, $B600–$B7FF for EEPROM, and $7C00–$7FFF for the Oki clock/calendar chip. To make things even simpler, the I/O and EEPROM blocks can be moved to the start of any 4K page, and the Oki clock and EEPROM blocks can be disabled altogether. Refer to the Motorola "68HC11 Reference Manual" (ISBN 0-13-566720-8, Prentice-Hall) for the necessary HC11 instruction set and register details.

Listing 3 is the equivalent of "Hello world!" for RTC-HC11 development; it simply blinks the RTC-HC1 l's LED to indicate that at least the MCU itself is alive. The program can be assembled with the AS11 assembler and then downloaded to the board using either the ROM monitor's "LOAD T" command or DLOAD . EXE, a quick and dirty download program for the IBM PC.

Another development option is to use one of the BUFFALO series of ROM monitors to download and test your code directly on the HC11. This monitor contains a built-in line assembler/disassembler so short test routines can be entered directly into RAM. Standard memory move/change/fill and EEPROM support is also included. The monitor itself resides in a single 8K EPROM and is installed in socket U8 on the RTC-HC11. Don't forget to install JP12 so the instruction trace and breakpoint commands will work.

```
32170 WHILE TB<>0
32180   IF TB>=LL THEN 32200
32190   LL=TB
32200   CK=0
32210   PRINT "S1";
32220   PRINT HEX2(LL+3);
32230   PRINT HEX(SA+OF);
32240   FOR X=0 TO LL-1
32250      SD=PEEK(SA+X)
32260      PRINT  HEX2 (SD) ;
32210      CK=CK+SD
32275      TB=TB-1
32280   NEXT X
32290   CK=CK+LL+3
32300   CK=CK+((SA+OF).AND.$ff)
32310   CK=CK+((((SA+OF).AND.$ff00)/256).AND.$ff)
32320   CK=CK.EOR.$ffff
32330   PRINT HEX2(CK.AND.$ff)
32340   SA=SAtLL
32350 ENDWH
32360 RETURN
```

listing P-continued

RTC-HC11 INITIAL SETUP

Setting up the RTC-HC11 is very simple. Any standard RS-232 serial terminal can be used as a console device for the RTC-HC11. Alternatively, an IBM PC using a communications program can be substituted. The console device you choose must be capable of being configured for 9600 baud, 8 data bits, 1 stop bit, and no parity if you will be using either the ROM monitor (BUFFALO 3.4), or BASIC-l 1. The console device is connected to the RTC-HC11 through J2.

Once everything has been connected and double checked, turn on the power to the terminal and allow it to warm up. Next, apply power to the RTC-HC11. If the ROM monitor is installed and everything is connected properly you should see the message:

```
BUFFALO 3.4 (ext)
>
```

Enter a "?"for a list of BUFFALO commands. If you're using BASIC-l 1, and everything is working properly, your display should look like this:

```
PORTD   EQU    $08
DDRD    EQU    $09
SPCR    EQU    $28
        ORG    $0000          ; Internal RAM
main:
        ldy    #$1000         ; base of on-chip I/O regs
        ldaa   #$04
        staa   SPCR,Y         ; set port D pin mode
        lds    #$ff           ; set top of stack
        ldaa   #$f0           ; set data direction register
        staa   DDRD,Y
top:
        ldb    #$05           ; timing loop
        bset   PORTD,Y $20    ; turn led ON
lp0:
        jsr    dly50          ; delay for half a second
        decb
        bne    lp0
        ldb    #$05
        bclr   PORTD,Y $20    ; turn led OFF
lpl:
        jsr    dly50
        decb
        bne    lpl
        jmp    top            ; repeat forever ...
***
* DLY50 - 50 ms delay subroutine
***
dly50:
        pshx
        ldx    #$411e
d50lp:
        dex
        bne    d50lp
        pulx
        rts
```

Listing 3—*LED blink test program.*

INSIDE THE MC68HC11A1

SINGLE-CHIP MODE-The MCU functions as a self-contained microcontroller and has no external address or data bus. This mode provides maximum useof the pins for on-chip peripheral functions, and all address and data activity occur within the MCU. This mode would not normally be used on the MC68HC11 Al, because of no internal ROM.

EXPANDED MULTIPLEXED MODE-The MCU can address up to 64K bytes of address space. Higher-orderaddress bits are output on the port B pins, and lower-order address bits and the data bus are multiplexed on the port C pins. The AS pin provides the control output used in demultiplexing the low-order address at port C. The R/W pin is used to control the direction of data transfer on port C bus.

BOOTSTRAP MODE-All vectors are fetched from the 192-byte on-chip bootloader ROM. This mode is very versatile and can be used for such functions as test and diagnostics on completed modules and for programming the EEPROM. The serial receive logic is initialized by software in the bootloader ROM, which provides program control for the serial communications interface (SCI) baud and word format. In this mode, a special control bit is configured that allows for self-testing of the MCU.

TEST MODE-Primarily intended for main production at time of manufacture; however it may be used to program calibration or personality data into the internal EEPROM. In this mode,a specialcontrol bit isconfigured to permit access to a number of special test control bits.

```
BASIC-11  Version  1.54
Copyright (c) 1986 -  1990
by Gordon Doughman
READY
#
```

The "#" character is the BASIC-l 1 prompt. Type "FREE" and press the "Return" key. BASIC-11 should respond with a decimal number.

MORE TO COME

This article reads a little like an equipment manual, but there aren't too many other ways to describe a hardware and software microcontroller design. I'd much rather be detailing the trials and tribulations of "remotecamera positioning is an unilluminated environment" or experimenting with "substrate-level particle acceleration caused bycollapsingelectromagnetic fields initiated through errant program execution."

In all seriousness, I have a few projects in mind where the RTC-HC11 is perfectly suited and I'll bring them to you as soon as they are working. Since this is my first 68xx project in a long time (my original Home Control System [HCS] used a 6802) there's a big hole to be filled.+

Special thanks to Gordon Doughman for his contributions to this project. Thanks also go to all those at Motorola who went out of their way to provide us with assistance.

Steve Ciarcia (pronounced "see-AR-see-AH") is an electronics engineer and computer consultant with experience in process control, digital design, and product development.

Burt Brown received a BS in Computer Sciencefrom California State University and has been working with computers and electronics for over ten years. In his free time, he enjoys bicycling, running, and Italian food.

The RTC-HC1 1 is available from Micromint. For pricing and data sheets, call (800) 635-3355 or fax (203) 872-2204.

IRS

409 Very Useful
4 10 Moderately Useful
411 Not Useful

FEATURE ARTICLE

Markus A. levy

Designing with Flash Memory

Is There a New Alternative to EPROM and SRAM?

Flash memory (in general) is capturing market share from other memory technologies. It is replacing EPROMs that were traditionally used for code storage because, along with equivalent nonvolatility, it also allows in-system updates. Battery-backed SRAMs that once were used for data acquisition, parameter storage, and even solid-state disks are now targets for the inherently nonvolatile and lower-cost flash memory devices. Many notebook computer OEMs conclude that low power, light weight, and reliability are most easily obtained with a completely solid-state machine, Flash memory has achieved a density ramp from 256K bits to 2 megabits in two years, Combined with a special flash file system from Microsoft, flash memory can even replace the mechanical disk drive.

With the design described in this article, you have a platform demonstrating flash memory's functionality and flexibility. Applications range from data acquisition through an I/O port, to a DOS-compatible, solid-state disk. But first, a few essentials.

EPROM AND BEYOND

Derived from an EPROM process base, Intel's ETOX-II flash memory technology has similar nonvolatility, reliability, and array densities. In fact, the flash memory cell is identical to the EPROM structure, except for the thinner gate (tunnel) oxide. This is where the similarities end. The thinner gate oxide enables flash memory to be erased and reprogrammed in-circuit, typically 100,000 times. The name "flash" is derived from its one-second chip-level erase and microsecond-level byte-write times versus the slower, millisecond-level byte-write times for conventional EEPROMs.

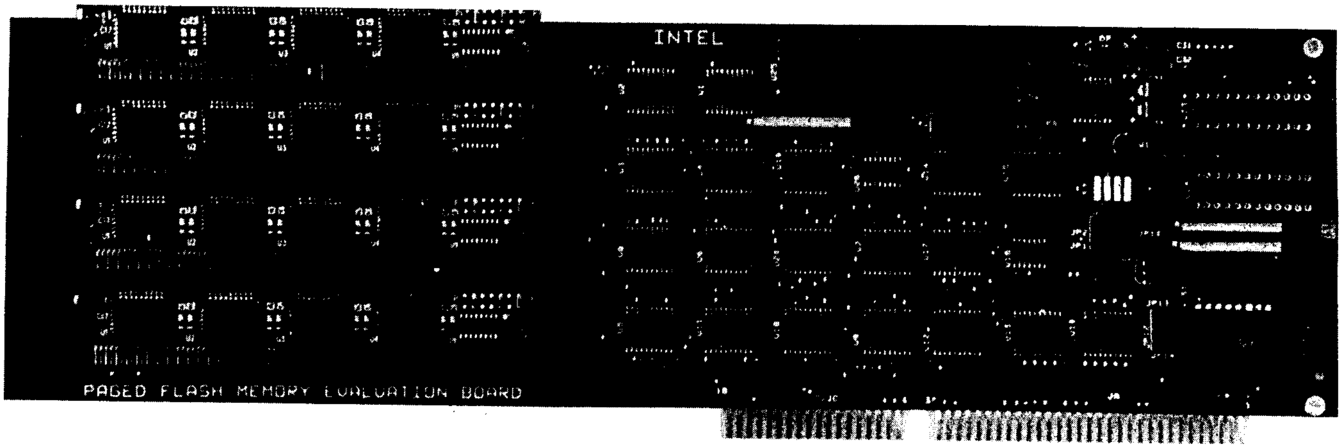
Flash memory devices have a command register architecture that providesamicroprocessor-compatible write interface. Erase, program, verifications, and other operations are initiated by issuing the proper command to the flash memory device. Twelve volts must be applied on V_{pp} for the command register to respond to writes and execute the operation. The 12V requirement doubles as an added security feature for data integrity. If you are familiar with other memory subsystems, designing with flash memory is as simple as any other technology.

In addition to discrete components, Intel offers flash memory in SIMM and IC memory card formats. This design will use these modules, so I've included some pertinent information. The 512K-byte x 16 Intel Flash SIMM (SM28F001AX) is based on an 80-pin JEDEC standard that accommodates density upgrades and presence detect (a hard-wired ID that indicates SIMM density and speed). The eight l-megabit flash memory devices on this module are paired up as high and low bytes. They are selected using the SIMM's write enable high and low (*WEH and *WEL) signals.

Intel's IC memory card adheres to the Personal Computer Memory Card International Association (PCMCIA) standard. This standard specifies physical, electrical, information structure, and data format characteristics of the card. Most impressive is the size, measuring 85.6 mm x 54.0 mm x 3.3 mm. Its 68-pin interface includes 26 address lines used to directly address 64 megabytes. All buffering and chip-level decoding is contained within the card, greatly simplifying the board-level design. Intel's flash memory card is available with one and four megabytes. These cards will continue to grow in density, becoming more and more competitive as disk drive replacements.

MEMORY METHODS

Three fundamental addressing methods can be implemented when interfacing a flash memory array to a system bus: linear, I/O, and paged. Each method has its benefits and drawbacks. A linearly addressed memory array is mapped directly into the sys-



tern's memory space and allows the highest performance. However, the memory array would be insufficien tly small in systems having limited memory space, as with the 8086. But this method is practical in an 80386 (or other 32-bit processor) family system with a large memory space available.

An I/O-mapped memory array uses one address-an I/O port-to transfer data. This method requires the least amount of system memory space but also yields the lowest performance.

A page-mapped memory system is a hybrid of these two approaches. It allowsaverylargememoryarray with a minimal system interface. A page is a moveable window into the total memory array. It selectively opens different portionsof thearray by writing a page number to the decoding circuitry. This page ranges in size from 8K to64K bytes. Analogous to a cache, a larger page size requires less frequent switching. Although switching pages represents a performance degradation, this can provide the optimal balance between performance and memory space availability within the system.

Our design is based on this page-mapped technique. A 64K-byte page size reduces the decoding circuitry. The PC/AT has been chosen as the execution platform, but with minor modifications to the control signals, any microprocessor environment can be used. Before beginning this design, it would be helpful to reacquaint yourself with the basics of the AT I/O channel bus.

The subsystems within this design (Figure 1) are the memory decode circuitry, I/O and its associated decode logic, and a 12V generator for V_{pp}. The Intel flash memory resides in four SIMMs. The board handles an upgrade path to 16M bytes, based on 4M-byte SIMMs.

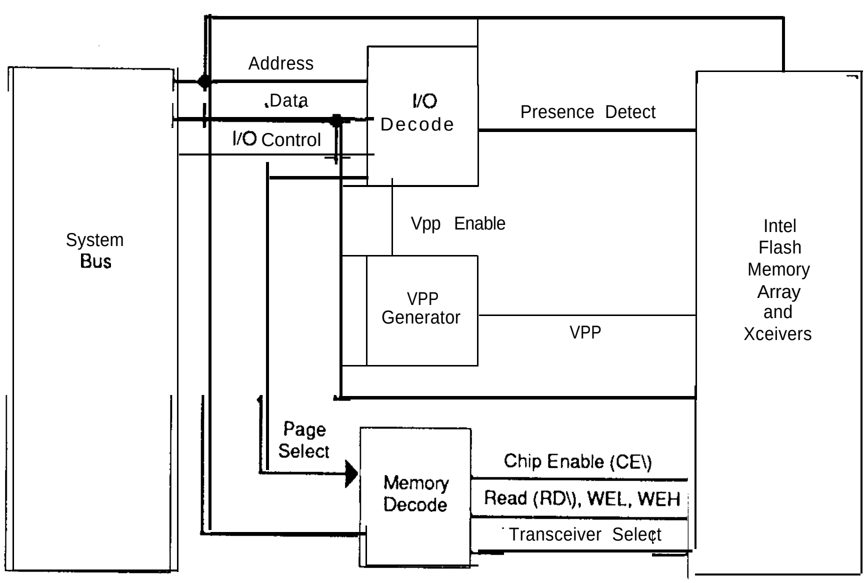


Figure 1 --The *subsystems in a flash memory board design include memory decode. I/O. and a 12V generator.*

ADDRESS DECODING

Flash memory addresses can be decoded in one of two ways: row-column and conventional decoding using separate chip enables. The row-column approach of Figure 2is appropriate if you are motivated to reduce board traces. In row-column decoding, rows are Output Enables (*OE), Write Lows (*WRL), and Write Highs (*WRH); and columns are Chip Enables (*CE). Although the SM28F001AX uses only four chip enables, eight are provided since four-megabyte SIMMs could consist of sixteen 2-megabit flash memory devices.

Page selection, discussed in more detail later, is accomplished by writing the page number through an B-bit I/O port to a latch. This will allow access to 256 64K-byte pages. Page signals, PO-P2, are directly connected to A15–A17 on each SIMM. They decode pages on the device level. The row-column signals are derived by decoding page signals P3–P7. They enable components on the SIMMs.

The row-column approach, however, suffers from simultaneous selec-

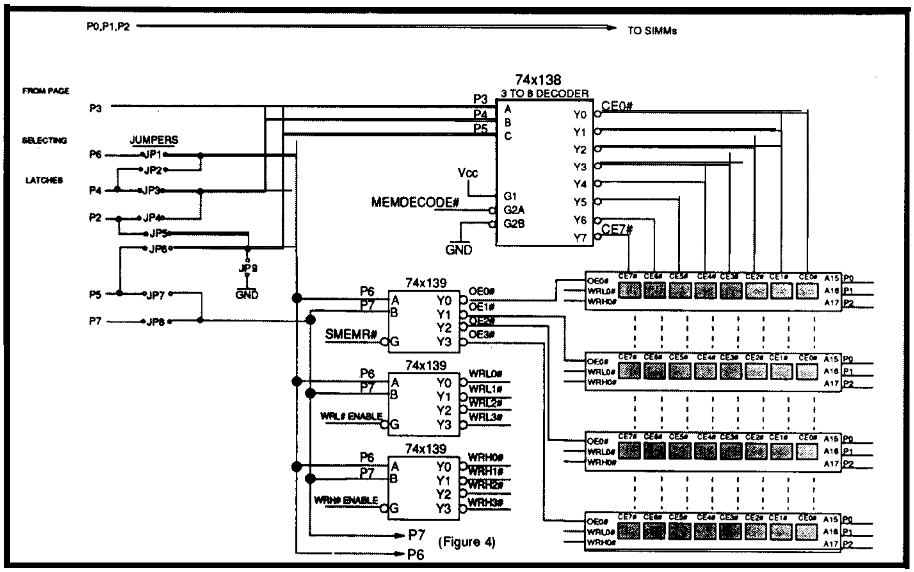


Figure 2-Row-column *addressing can be used to reduce board traces.*

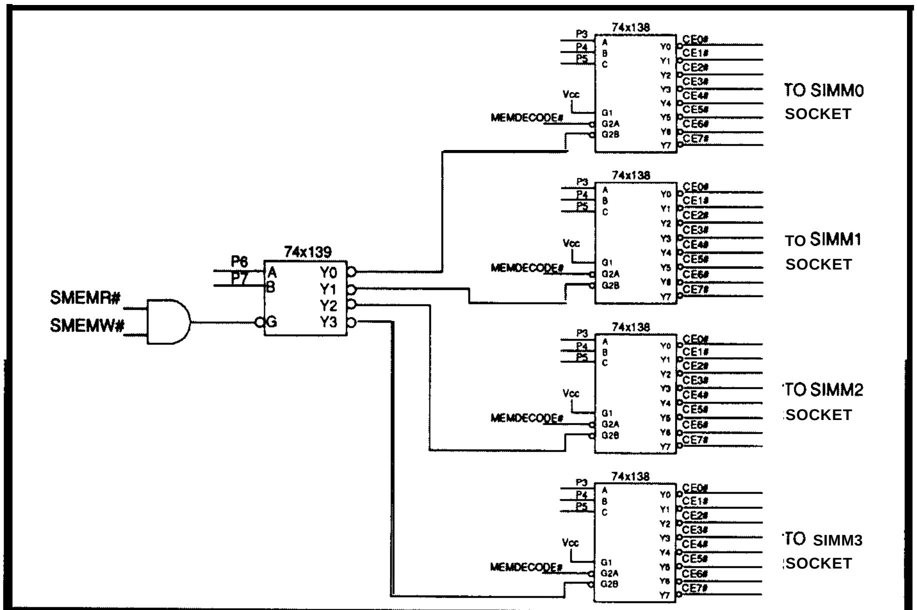


Figure **3—***Using separate chip enables trades off board complexity for* **low** *power usage.*

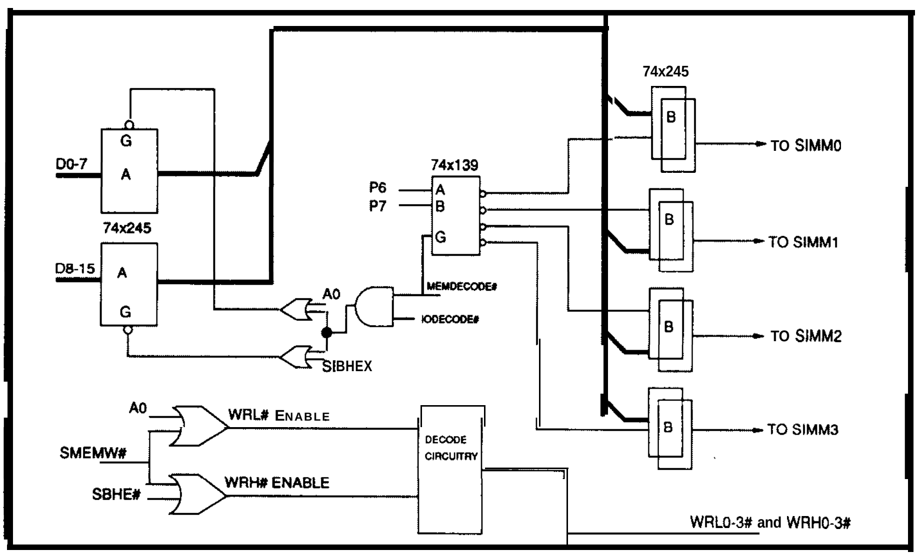


Figure 4-A *simple buffering scheme is used to connect the memory to the system bus.*

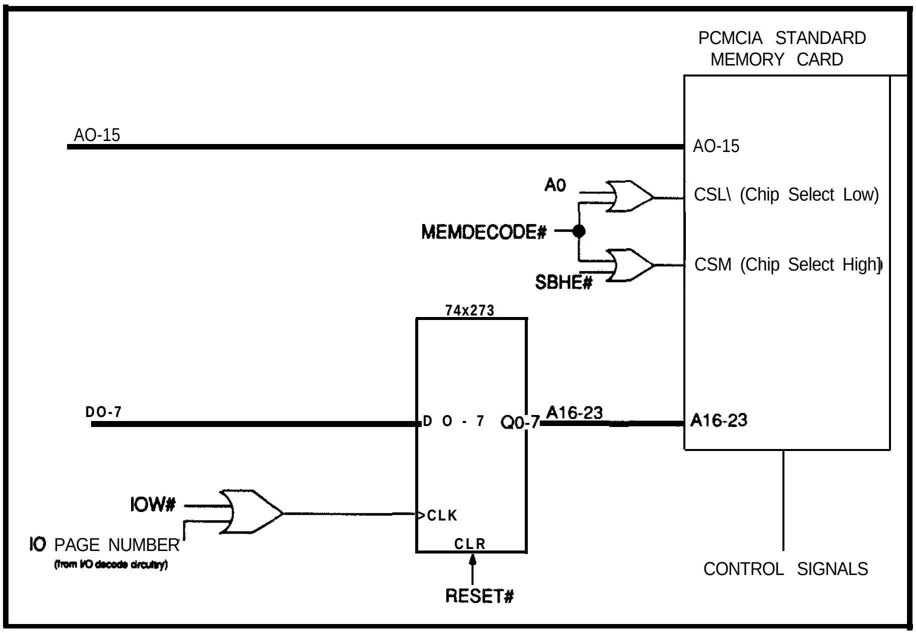


Figure 5—*Moving some buffering onto a separare memory cara reduces the amount c necessary interface hardware.*

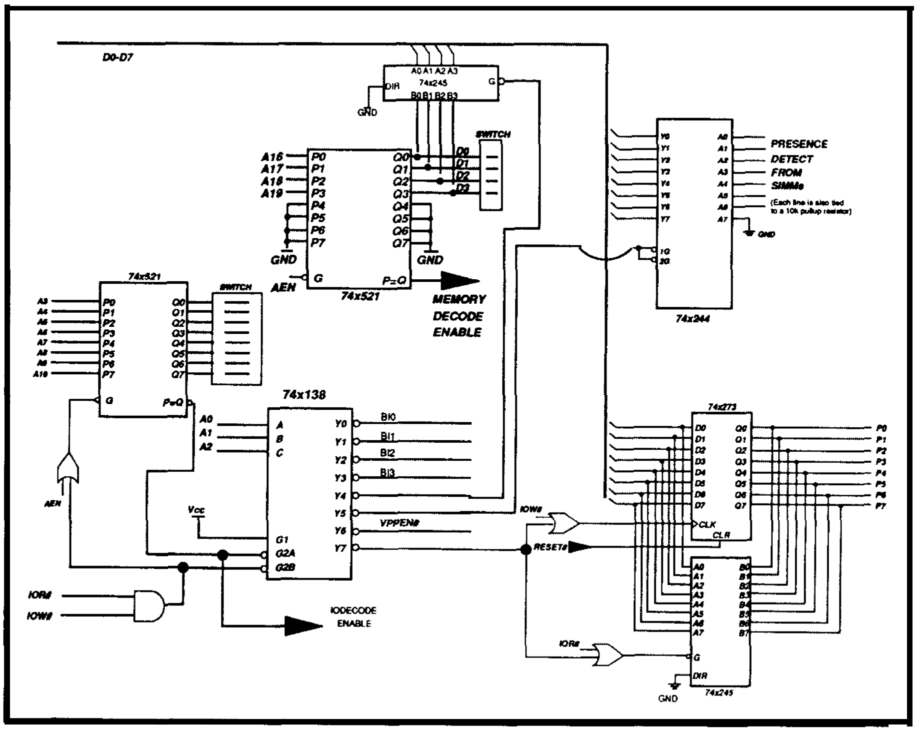


Figure 6—*DIP switches are used to set the board's address in the PC's I/O space.*

tion of eight parallel device chip enables. The conventional, separate chip enable decoding method has lower power consumption. Using this method (Figure 3), upper page signals, I'6 and P7, decode which SIMM is selected.

Page selection becomes more complicated when accounting for density upgrades. To understand this there are two things to keep in mind: each SIMM socket handles a maximum of 64 pages, (i.e., 4-megabyte SIMMs), and Al 7 is used on a 2-megabitdevicebutisa"NoConnect"onthe l-megabit part. A "No Connect" pin implies that page selection will not be contiguous with SIMMs less than four megabytes in size. Accommodating noncontiguous pages increases software complexity. Regardless of the page decoding method, a jumper scheme rearranges the page signals and accommodatesdensity upgrades.

According to Figure 2, the jumper settings are as follows:

1-MByte SIMM-J7, J2, J9, J4
2-MByte SIMM (16 x 28 F010)—J7, J1, J3, J5
2-MByte SIMM (8 x 28 F020)—J7, J1, J9, J3
4-MByte SIMM (16 x 28 F020)—J8, J1, J6, J3

Figure 4 shows the buffering required for system bus interfacing. The PC I/O channel bus is limited to two TTL loads on any one line. The "A" transceivers are connected directly to the I/O channel bus. Additionally, each SIMM has its own pair of "B" transceivers to reduce capacitive loading that results from tying more than eight flash memory devices together.

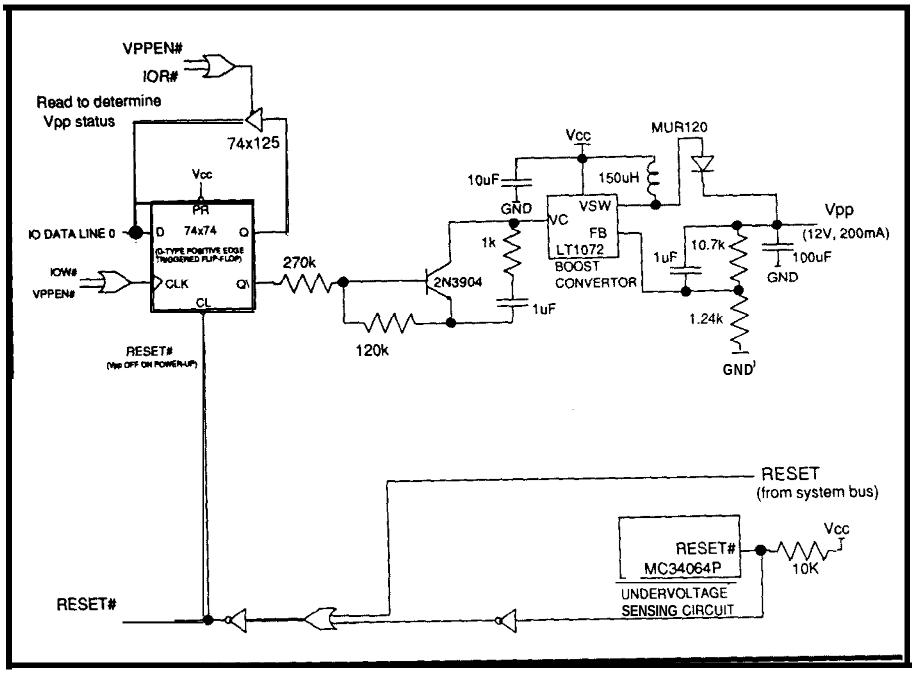


Figure 7-A ***regulated boost*** converter **made from *Linear*** Technology's *LT1072* supplies the *12 V necessary for programming.*

SIMPLIFIED DECODING HARDWARE

You are not limited to using SIMMs in your design. The same basic techniques will work for discrete components or other module types, such as memory cards. The IC memory card provides the simplest solution with its integrated decoding. Writing the page number through an I/O port to a latch accomplishes the same page selection as before, but now the "data" translates directly into memory card address inputs A16–A23. This eliminates the entire decoding structure shown in Figure 2 or 3. The IC card itself also contains the "B" transceiver buffering. This results in the hardware reduction shown in Figure 5. You will want to purchase a spring-loaded connector from Fujitsu, AMP,

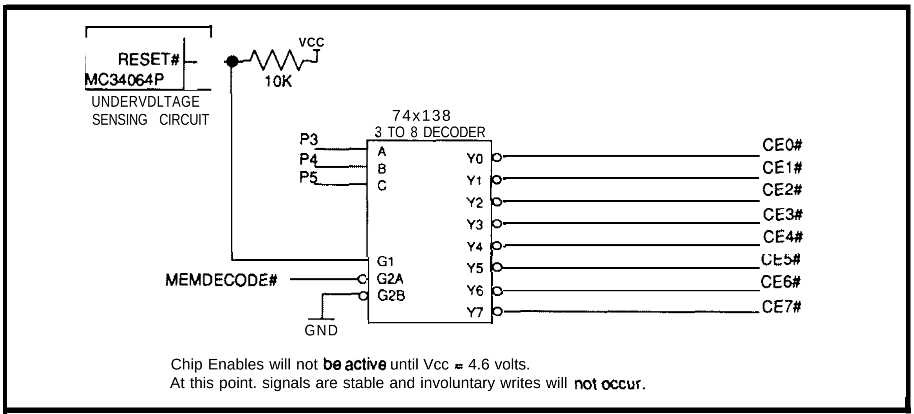


Figure 8—*An MC34064P undervoltage sensor is used to disable all writes to* memory *when the power starts to fail.*

or ITT Cannon to use with the memory card, laying out your board so that the memory card can be retrieved out the back of your PC. Pushing the button on the connector ejects the memory card for data security or transport.

I/O PORTS ON THE PAGE MEMORY BOARD

The page-selecting I/O port mentioned is one of eight ports in this page memory board design. The 74x521 comparator's inputs begin with A3 to simplify the decoding circuitry for the I/O port addresses. This places the base I/O port on an even 8-byte boundary. Use discretion when setting the switch to avoid conflict in the AT environment. I recommend using I/O addresses between 300H and 318H because this area is assigned for prototype cards.

Listed in the order in which they appear on the 74x138 decoder in Figure 6, the eight I/O ports in this design are: BIO–BI3 (accesses the board's identifiers); the window address within the system's memory space; the SIMM's presence detect; the V_{pp} enabling register; and the page number reading and writing ports.

I didn't diagram the board's identifier circuitry because the implementation is extremely simple. It consists of four identical units made up of a 74x244 and an 8-input DIP switch. The four enables, BIO-B13, from the I/O decoder connect to the corresponding unit's **enable.** The I/O space is scanned for the identifiers to locate the board.

The user-selectable page window address can be placed on any 64K-byte boundary within the DOS 1-megabyte range, but only the adapter ROM area between COOOOH and EOOOOH will alleviate compatibility problems. Using additional inputs on the "Memory Decode Enable" 74x521 and an 8-input DIP switch allows window placement above 1M byte. Simply connect address lines A20–A23 to input pins P4–P7, respectively. The Presence Detect (PD) pins from all four SIMMs are wire ORed together into the 74x244. This eliminates having to read each SIMM's I'D pins separately, but SIMMs must be installed with equivalent density and speed.

The page number is written and read through the same I/O port. Minimal circuitry ensures that the system powers up at page zero. The page memory board's *RESET signal connected to the CLR input of the 74x273 latch accomplishes this task.

V_{pp} GENERATION

Why is V_{pp} generation necessary on this flash memory board when the AT I/O channel already has a 12-V supply? True IBM-compatible PCs specify the necessary ±5% 12-V supply tolerance. However, some systems have wider 12-V supply tolerances. Local 12V generation via the circuitry described below ensures fast, reliable flash memory operation.

Linear Technology's LT1072 switching regulator, a 5-V-to-12-V feedback regulated boost convertor, is the heart of the V_{pp} generator (Figure 7). A 100-µF capacitor at the output handles up to 200 mA, necessary for software that programs or erases

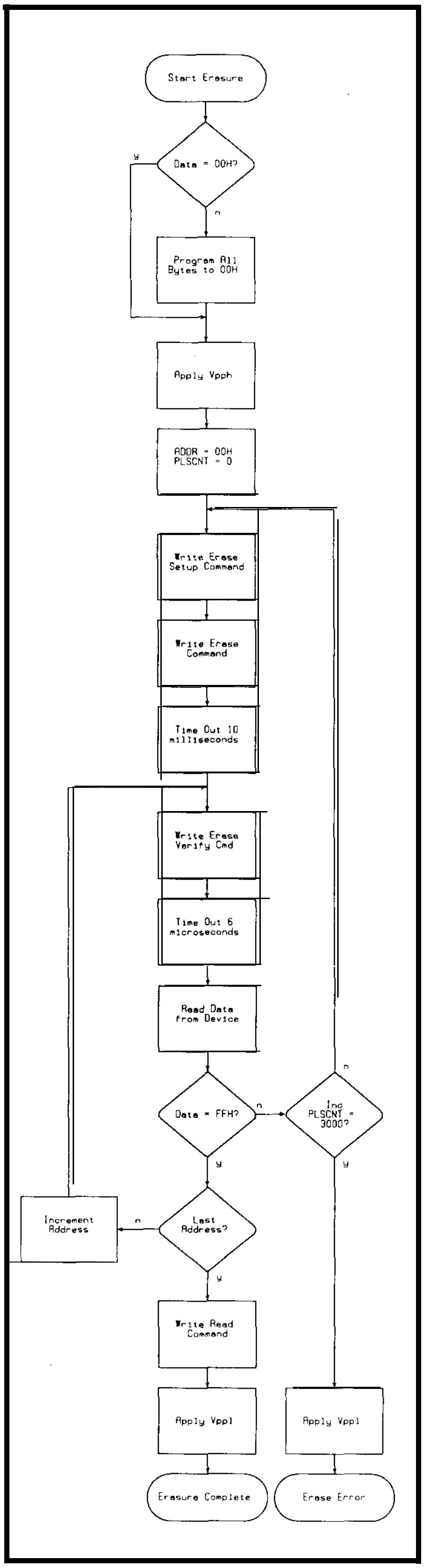


Figure 9—*The flash erase algorithm.*

eight flash devices simultaneously. Turning V_{pp} off when not in use conserves power, but this capacitance value requires approximately 100 ms to fully charge to 12 V. Reducing the output capacitance value and limiting the number of flash memory components accessed simultaneously decreases the ramp time. The diode, MUR120, prevents inductor current absorption from the charged output capacitor. During system power-up, spurious noise may generate writes which are actually the sequence of flash memory commands that initiate erasure or programming. Disabling V_{pp} until voltages stabilize provides power-up protection. The Motorola MC34064P is an undervoltage sensing circuit that begins functioning when V_{cc} is above 1 volt. Between 1 and 4.6 volts, the MC34064P's *RESET output or AT system RESET clears the 74x74. While the 74x74 remains cleared (or *Q = 1), the 2N3904 is on, the VC input of the LT1072 is 0 volts, and the VOLTAGE SWITCH (VSW) output is off. Writing a one to the V_{pp} enable latch forces *Q low, turning off the transistor. This puts the VC input at 5 V, and VSW output generates 12 V.

You do not need the circuitry just described if your system's 12-V supply meets the V_{pp} specifications. However, because software may accidentally (or coincidentally) generate a valid flash memory command to a flash **memory address,** install a switch **to turn off V_{pp} when not in use. A low-**resistance&T (Motorola MTD4P05) performs this duty.

The possibility of spurious writes to the flash memory devices during power-up still exists. Again, the same undervoltage sensor (MC34064) solves this problem. The *RESET output becomes the 74x138 decoder's active-high enable. This controls the chip or write enables for the flash memory devices (Figure 8).

A FEW ADDITIONAL POINTERS

Ground *MEMCS16 so the PC/AT recognizes your board with a 16-bit bus width. The original design operated in both 8- and 16-bit systems. This flexibility is accomplished with additional decoding that multiplexes the high data bus onto the lower data bus. Again, the IC memory card automatically conforms to either bus width because the extra decoding is handled internally.

As with any circuit design, it is important to follow good design principles. For example: decouple power supplies with 0.1-µF capacitors between V_{cc} and V_{ss} of every device; and short board traces help minimize noise.

SOFTWARE

Hardware without software is like a computer without a processor. Therefore, understanding program and erase algorithms is the first step towards functional flash memory. Recall from our earlier discussion that operations on flash memory are software controlled using the internal command register architecture. I have included the complete algorithms (Figures 9 and 10). *[Editor's Note: Software for this article is available on the Circuit Cellar BBS and Software On Disk #18. See page 92 for downloading and ordering information.]* After working the Intel flash memory hotline, I would like to discuss the common mistakes.

The best piece of advice that I can give is *please* follow the algorithms. Many people unsuccessfully try their own custom versions. There are no cutting comers. Starting with the basics, let me elaborate a few points about the algorithms:

1) Flash is programmed to a binary zero by adding charge to the transistor's floating gate. Contrarily, charge removal erases the cell to a binary one. During the erase operation, the device is "flashed" as charge is simultaneously and equally pulled off the floating gate of every memory cell. The device must be preprogrammed to all zeros before erasure so an already erased cell is not further depleted of charge.

2) Prior to writing any command, switch on V_{pp} and allow ample ramp time for proper operation. Dropping V_{pp} below 7.5 volts from within any operation, places the device in the read mode. Similarly, abort an operation by issuing two consecutive reset commands (FFH) followed by the read command (00H).

3) Closely observe delay times to achieve the highest performance and

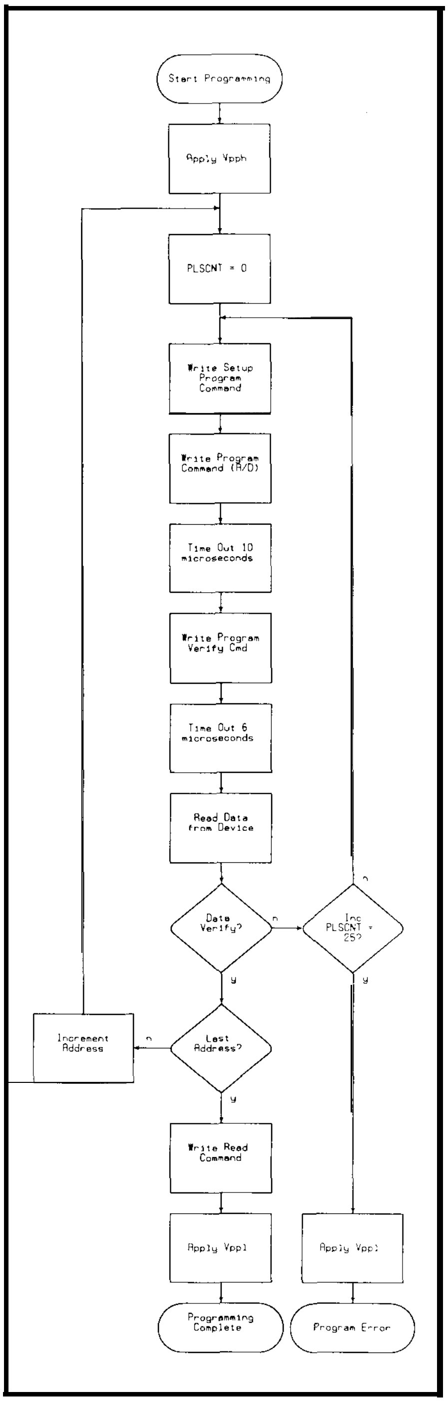


Figure 10—*The flash programming algorithm.*

reliability. Use the STI instruction in the software drivers to avoid system interrupts during these delays. Execute CLI once the corresponding verify command is issued.

4) The verify operation internally creates marginal conditions to ensure accurate and reliable results. The 6-µs slew time delay following the verify command allows the margin voltages to settle. In the verify mode, programmed data is guaranteed to be "permanent" when it matches the data being programmed.

USING YOUR PAGE MEMORY BOARD AS A DATA RECORDER

There is a nearly endless list of data recording applications, including digital imaging, digital photography, point-of-sale terminals, patient monitors, and flight recorders. The page memory board is appropriate for many data recording applications where an I/O port of the PC/AT accumulates data.

The programming algorithm demonstrates the byte programming capability of flash memory. In other words, once the device is erased, bytes reprogram randomly. In some recording applications, data is received in packets, A pointer determines where to begin programming the next free location within the flash memory.

Interleaving increases write performance by using the idle time during the 10-µs program delay. Addresses are offset such that each successive data byte gets written in different devices, looping back in time to issue the verify command. Reading the data back would be done in a similar fashion.

Word-wise, or parallel, programming of two devices provides an additional means of increasing write performance. Note that flash memory devices may program at different rates. Therefore, the original algorithm must be modified during the verify operation. If only one byte of the word has verified, the program command and data are sent again to the unverified byte. Mask the command sent to the device that verified to maintain word-wise programming. A mask is the substitution of a reset command for the program and verify commands. That way, the programmed bytes do not get further programmed on subsequent pulses.

The page memory board will perform these software techniques. However, first write the software that determines the location and capacity of the board. First, scan the I/O space for the board's identifier. The location of the first identifier byte is also the base address for the eight I/O ports. Using the proper offset, read the I/O port that enables the base-memory address transceiver. For SIMMs, calculate the memory capacity by first reading the Presence Detect pins, followed by reading the individual flash memory device identifiers. Alternatively, read the Card Information Structure in the PCMCIA standard memory card for the capacity.

The preceding steps will confirm the basic functionality of your hardware. Practice programming the flash devices with data from a RAM-based array. For example:

```
; Software to read in ASCII test
; pattern to program into flash
DATA ARRAY SEGMENT
STORE_IN FLASH
    DB 'ASCII test pattern to
    DB 'be stored in flash'
DATA_ARRAY ENDS

CODE SEGMENT
    mov  ax,DATA_ARRAY
    mov  ds,ax
    mov  si,0
    mov  cx,size STORE_IN_FLASH

more_input:
    mov  al, [si]
    call FLASH_PROGRAM
    inc  si
    loop more-input
```

Now the fun begins. Once you've put your Flash Paged Memory Board together and tried it out, imagine converting it into a solid-state disk. It's beendoneusingsoftwaredriversfrom Microsoft. In an upcoming article, we will discuss the structure of the Microsoft Flash File System and show you how to interface it to your board.

Finally, I would strongly suggest thatbeforeattemptingthisdesign,you obtain the appropriate flash memory literature from Intel (see the source box).

SOURCE

Intel Literature: (800) 5484725

Data sheets

SM28F001AX	1 M-byte SIMM	#290244
1MC001FL	1 M-byte IC memory card	#290399
4MC001FL	4 M-byte IC memory card	#290388
28F020	2 M-bit device	#290245

Application notes

AP325, Guide to Flash Memory Reprogramming	#292059
AP343, High-Density Applications Using Intel Flash Memory	#292079

Markus Levy is an application engineer at Intel Corporation in Folsom, California, and holds a B.S.E.E from California State University. Hisspecialties includesoftwareand hardware implementations of solid-state disks in portable computers. His favorite away-from-work pastimes include home remodeling, swimming, and parenting.

IRS

4 12 Very Useful
413 Moderately Useful
414 Not Useful

ANSI Controls and Fixed Points

The Furnace Firmware Project Continues

FIRMWARE FURNACE

Ed Nisley

Last winter when I laid out the Furnace Firmware project I forgot about Serious Heat and Humidity: right now it's 90°F at 90% and I have no intention of firing up the furnace for a code check! So this column includes the ANSI LCD driver and a fixed-point math package. Never fear, when the heating season rolls around again you will get the final wrapup!

PARALLEL CHARACTERS

The Furnace code so far includes keypad input and LCD output routines, but they are entirely distinct from the normal C console I/O routines. Wouldn't It Be Nice If the C `getch()` function returned characters from the keypad and `putch()` sent output to the LCD panel, in addition to the normal C serial console support?

The keypad firmware in CIRCUIT CELLAR INK #16 produces a character for each key press, using the timer tick to control polling. Listing 1 shows the few lines of code needed to splice those characters into `getch()`. A similar addition to `kbhit()` (which is not shown here) indicates when a character is available from the keypad. The `PadConsole` bit controls the new code in each routine, so you can disable the connection and process keypad characters separately by calling `KbdGetChar()` when needed.

The keypad translation table now includes the CR, ESC, Ctrl-C, BS, and hyphen characters needed for "normal" console input. Because the three remaining function keys still return characters above 80 hex, `getch()` no longer clears bit 7 as it did for characters from the serial port. The piezo beeper sounds off only when characters come from the keypad.

On the other side of the user interface, sending a character to the LCD firmware as well as the serial port is so simple we don't even need a listing to explainit: if the `LCDEcho` controlbit is set, just call `LCDSendChar` from `putch()` after sending the character to the serial port. As you saw in CIRCUIT CELLAR INK #15, `LCDSendChar` handles the common "control" characters: LF, CR, FF, and BS. As long as your application can use TTY-style output, you're in good shape.

However, you probably want to treat the LCD panel as a static display screen: write some fixed information, then move the cursor around to update selected values. This improves output speed because fewer characters are sent and, depending on the LCD controller, may also eliminate display glitches. But, because TTY-style output can't handle cursor positioning, this trick is impossible. With the `putch()` interface so far, you

```
_getch    PROC
          PUBLIC _getch

<<< startup code omitted >>>

;--- check for a char from the serial port

L?wait    MOV    A,RRing_cLevel
          JNZ    L?fetch

;--- check for keypad char

          JNB    _PadConsole,L?wait  ; skip pad if not enabled
          CALL   KbdTestChar
          JNC    L?wait              ; none, so continue waiting
          CALL   KbdGetChar          ; have one, get it
          SJMP   L?polish            ; fix it up

;--- get char from serial port ring buffer

L?fetch

<<< serial port code omitted >>>

;--- polish the character

L?polish  CJNE   A,#CR,L?1           ; use LF instead of CR
          MOV    A,#LF
L?1
          %IF    0
          ANL    A,#$7F              ; strip high bit
          %ENDIF
          MOV    R3,A
          MOV    R2,#0               ; high byte always zero
          RET

_getch    ENDPROC
```

Listing 1 -Combining *keypad input with the serial port requires only a few lines of code. If the keypad routine has a character ready, it will take precedence over one from the serial port* ring buffers.

Screen coordinates start with Row 1 and Column 1 in the upper left column. The effect of coordinates outside valid screen boundaries is undefined.

Cursor Control

```
ESC[r;cH   Move cursor to specified row and column
ESC[r;cf   ditto, this is a synonym

ESC[nA     Move cursor up n rows, stop at top row
ESC[nB     Move cursor down n rows, stop at bottom row
ESC[nC     Move cursor right n columns, stop at right column
ESC[nD     Move cursor left n columns, stop at left column

ESC[s      Save cursor location
ESC[u      Restore cursor location

ESC[2J     Erase display and move cursor to upper left
ESC[K      Erase line from cursor to right edge
```

Display Attribute Control

Once set, an attribute applies until it is changed. Some codes set multiple display attributes.

```
ESC[xm     Set display attribute to x
ESC[x;ym   (You can set several attributes at once)
```

The most useful "x" parameters are:

```
0       Cancel all attributes (return to white on black)
1       Bold or bright
2       Dim or normal
5       Blink (only on displays that support blinking!)
1       Reverse video (black chars on white background)

30-37   Set foreground color
40-47   Set background color
```

Color codes are:

```
30/40   black       34/44   blue
31/41   red         35/45   magenta
32/42   green       36/46   cyan
33/43   yellow      37/47   white
```

Figure 1—*These ANSI control sequences allow a remote device to control the cursor location and display colors on a local terminal emulator. The ESC symbol designates the Escape character, ASCII code 27 decimal or 1B hex. There are no spaces in the sequences, and the last letter must be capitalized as shown.*

rewrite the whole smash from the top every single time. Something Must Be Done!

ANSI ESCAPEES

Fortunately, the American National Standards Institute (ANSI) has defined a set of cursor, color, and screen control codes that are (loosely) implemented in the PC world. Figure 1 shows the most useful codes; while the LCD can't display colors, it should at least gracefully ignore those codes. Most PC terminal emulators have an "ANSI BBS" mode in their **bag** of tricks, so you can probably use these codes with no trouble.

Notice that these control sequences start with an escape character (1B hex, 27 decimal, yclept ASCII 27), hence their common name of "ANSI **Escape Sequences."** A left-hand square bracket follows the ESC, which identifies this group of commands. Some commands have one or more numeric parameters and all end with an upper- or lower-case letter (and the case is significant!).

For example, the sequence to set the cursor to row 2, column 3 looks like "`ESC [2;3H`". The sequence "`ESC [1;33;41m`" changes the color of allsubsequent characters to bright yellow on a red background, at least on a color display. (Remeber that ESC represents the escape character.) The numbers are simple ASCII, not binary codes: the color-change sequence is ten characters long, not eight.

An ANSI driver intercepts all characters going to the display. It passes through all "normal" characters unchanged, but interprets the ANSI control sequences and takes whatever action is appropriate; the control sequences are not displayed on the screen. Under DOS the ANSI console driver is a separate chunk of software installed as a device driver, while under OS/2 you get ANSI controls "free" with every session.

Although you could write spaghetti to decipher the sequences, Figure 2 shows a state diagram that may make more sense and certainly results in better code. When each character arrives, the action you take depends on which state you're in: If the character doesn't match any of the actions for the current state, you reset everything and send the character to the LCD display. Remember that the states themselves don't do anything: they justremember"whereyou were" until the next character comes in.

The state machine driver code is too bulky to fit here, but there is one interesting aspect that bears mentioning. Most state machines you'll see are based on a table of "current state versus input character" that determines which routine gets control. Because the ANSI sequences have a fairly restricted syntax, I figured out how to trigger the code based on either a specific character (the equal sign) or a general class of character (any digit). Take a look at `ANSI.A51` for this column to get the details; it may simplify some of your own code.

The driver stores all characters in a buffer, so the action routines triggered by the final letter can validate their parameters. Because the Set Cursor Position and Display Attribute sequences have several numeric parameters separated by semicolons, I included a "semicolon detector" action to extract each preceding number

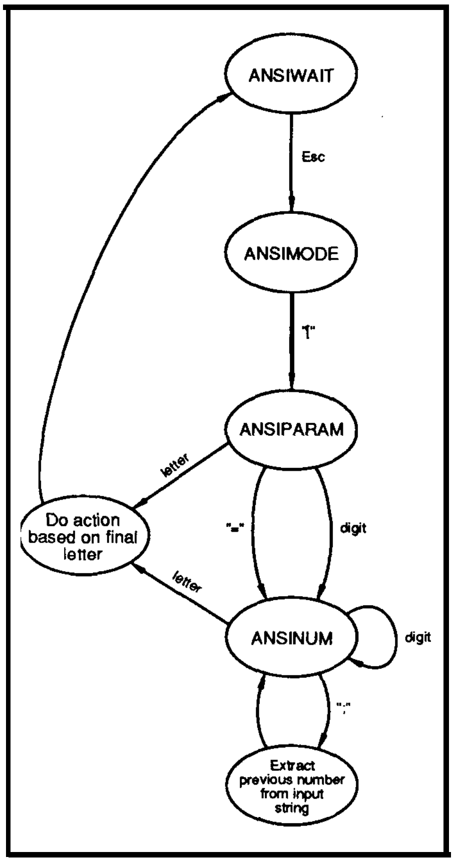


Figure 2—*This state machine decodes the ANSI control sequences shown in Figure 1. The final letter in each sequence determines the function.*

from the buffer into a numeric array. When the terminating letter arrives, the decoder action routine can access the saved parameters from the array.

A side benefit of buffering all the characters comes when the ANSI driver receives a defective sequence. Rather than just ditching the characters, the driver dumps the buffer to the LCD panel so you can see what went wrong. This simplifies debugging, and also permits programs to display "bare" escape characters (which show up as a left arrow). Not all ANSI drivers support this function, so you can't depend on it!

The **CDEMOANS** programexercises the ANSI driver by sending a series of test patterns to both the console and LCD. The two displays should match, save that your color monitor will have colored characters for some sequences. *[Editor's Note: Software for this article is available on the Circuit Cellar* BBS and Software On *Disk #18. See page 92 for downloading and ordering information.1*

Photos 1 and 2 show how a faked status display looks on both displays; while the PC console is more attractive, the LCD panel gets the job done.

If you look closely at those photos, though, you'll see something suspicious: where did those decimal points come from?

FRACTIONAL VALUES

Integer arithmetic is **good** enough for nearly all microcontroller applications, but sometimes you really need fractions. For example, a stepper motor might move 7.5 degrees per step and drive a slider 0.05 inches each time. You could pick tenth-degree units in one case and 10-mil steps in the other, but Wouldn't It Be Nice If you didn't have to write special arithmetic and display routines for each situation?

The obvious solution is to use floating-point numbers. The catch is that floating-point arithmetic is expensive in terms of both code size and execution speed. Microcontrollers don't have numeric coprocessors (by definition, I think), so all calculations must be done in software (or, if you please, firmware). The canonical "Hello, world!" occupies 5903 bytes using the Avocet fixed-point library; the floating-point version requires 16008 bytes and doesn't include any floating-point operations!

To put this in perspective, the Firmware Furnace routines don't use `printf ()` or the floating-point libraries. As a result, the keypad, LCD,clock, NVRAM, and C utility routines presented so far weigh in at about 8K bytes. There are some good reasons why I'm not enthusiastic about floating-point arithmetic.. .

```
iiiiiiii.ffffffff
        └─eight fraction bits
 └─ eight integer bits
```

Numbers are represented in two's complement form:

0.0 = 00 00	0.0039 = 00 01
1.0 = 01 00	-0.0039 = FF FF
-1.0 = FF 00	127.9961 = 7F FF
0.5 = 00 80	-128.0 = 80 00
-0.5 = FF 80	

Figure 3—*The fixed-point numeric format used in the Firmware Furnace project allows for numbers between - 128 and + 128. with a resolution of 0.39%. Each number occupies two bytes.*

The Firmware Furnace code must display some fractional values: temperatures (at least with the 10-bit ADC!), flow rates, time intervals, and so forth. I've put together a simple fixed-point math package that illustrates how to design a special-purpose numeric format.

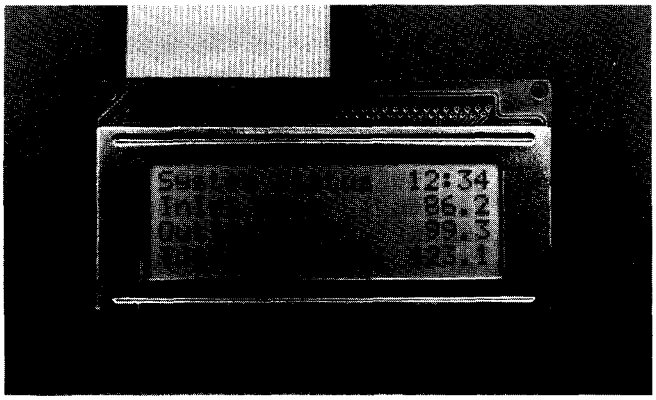


Photo 1 - A *small 4 x 20 LCD* display *works nicely to display all the* necessary **information.**

FLEXING THE FIX

In CIRCUIT **CELLAR INK** #9 I described the fixed-point numeric format used in the Mandelbrot Engine, which was an array processor dedicated to evaluating the Mandelbrot set formula. Because the calculations required high precision, the numbers only ranged from +8.0 to -8.0, but with a resolution of about 10^{-18}.

More mundane applications usually require a wider dynamic range and fewer decimal places. For example, domestic temperatures are under 100 degrees, furnace circulator pumps run at a few gallons per minute, elapsed times are a few hours, minutes, or seconds, and so forth. Generally, a range of a few hundred units and resolution of about 1% is entirely adequate, although your application may need more bits on one end or the other.

Figure 3 shows a fixed-point numeric format suited to the Furnace Firmware project. Values range from +127.9961 to -128.0000, with a resolution of 0.39% (or 1 part in 256). Each number requires two bytes, with an implied binary point dividing the number into eight integer bits and eight fraction bits. Remember that the point doesn't occupy any storage.

Although there is a binary point in the middle, the numbers add and subtract just like ordinary integers. Figure 4a shows how they combine; if this looks like third grade all over again, that's how simple it really is. Notice that results beyond the valid 128 range cause overflow because there are not enough bits to encode the full integer value; this is a drawback of fixed-point numbers in general and not the fault of this particular representation.

Multiplying two fixed-point numbers is a little bit trickier, because the product contains twice as many bits. Figure 4b shows that ordinary multiplication produces the correct result, but the binary point is located in the middle of the four-byte result. I used unsigned long int variables (which are 32 bits in the Avocet C

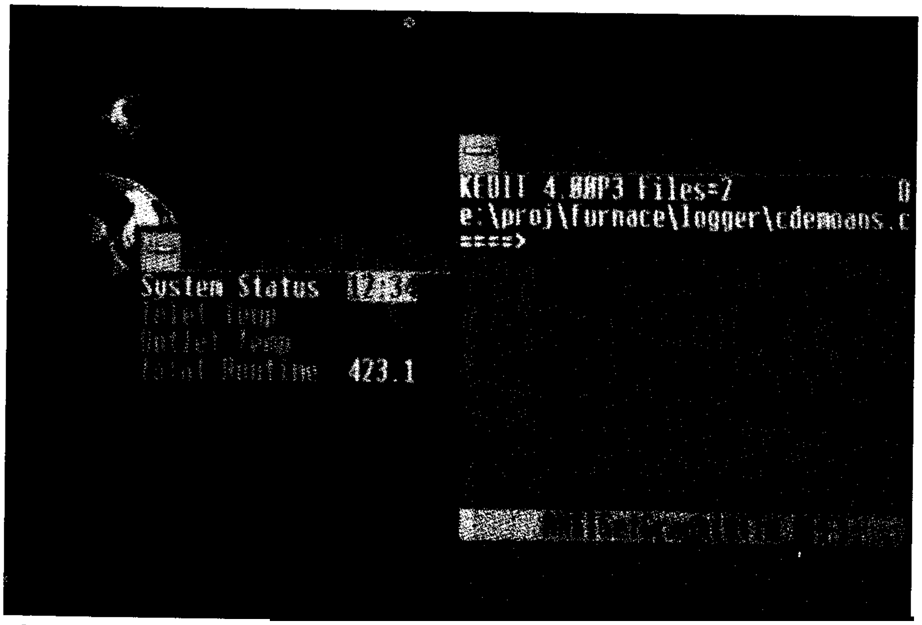


Photo 2—*The same display as in Photo 1 looks much nicer on a full-color CRT, but isn't quite as convenient.*

implementation) to hold the intermediate results and product. Extracting the result is a simple matter of shifting the product eight bits to the right and returning the low-order two bytes.

Division, being the inverse of multiplication, requires more setup. Figure 4c shows how the dividend is aligned to the high end of a four-byte unsigned long int and the divisor shifted into the middle of another. This produces the result directly in the lower two bytes of the quotient. An unshifted divisor would put the result in the middle of the quotient, thus needing a shift after the division.

In both routines, I calculate the signs separately and perform the long arithmetic on unsigned quantities to avoid obscure problems with C's sign extension and bit shifting logic. I'm pretty sure the code can be tuned up! Whatever you do, test your code at the ugly boundary conditions: I thought I had this stuff working quite a few times before it reached this condition.

Displaying a fixed-point number involves two steps because the integer and fraction parts must be converted separately. The former is easy enough; see `ConShowInt` in `CONSOLE.C51` for details. Converting the fraction to ASCII is ordinarily a tedious operation involving long division and considerable hocus-pocus. Listing 2 shows a shortcut: entries in a table relate fraction bit locations and the corresponding packed BCD equivalent, accurate to four decimal places.

The code in Listing 2 tests each bit position and accumulates a BCD total

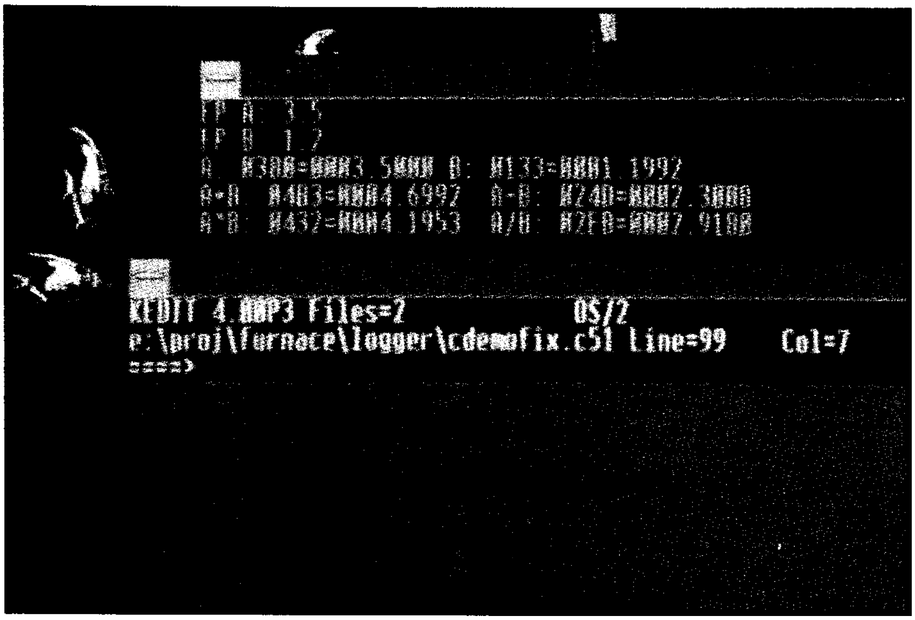


Photo 3—*The output* of the *CDEMOFIX* ***program shows the results of all four basic arithmetic operations on two user-entered numbers.***

for all the "one" bits. The final result is a fourdigit packed BCD value representing all eight fraction bits. Display is then just a matter of calling **a** routine that converts a two-byte binary value into an ASCII hex string. Think about it: there is no way to tell whether 1234 is hex or packed BCD. Slick, eh?

Besides, I've always wanted to use the Decimal Adjust instruction.. .

The CDEMOFIX program runs through several test sequences, then prompts you for a pair of fixed-point numbers and displays the results of all four arithmetic operations. Photo 3 shows the results on a PC monitor (sorry, they don't fit on the LCD panel!). The source code includes several handy routines, including input and display functions for both integer and fixed-point numbers.

If your controller application requires transcendental functions or other exotica, you'll be well-advised to use the standard floating-point libraries and functions instead of writing your own code. However, if your calculations are more along the line of scaling, offsetting, and displaying reasonable numbers, using a **fixed**-point math package will give you a substantial performance boost and a significant size reduction . ..benefits not to be sniffed at!

You will probably find that the dynamic range of ±128 available with a single integer byte is not enough. The next logical step is two integer bytes and one fraction byte, giving you ±32K with 0.39% resolution. If you can afford four bytes, add eight more fraction bits and extend the resolution to 15 parts per million. The algorithms I've used scale nicely to more bits and digits, although you will have to modify the register usage because I avoided buffering intermediate results in RAM.

```
(a)  1.0 t 1.0            0100 t 0100 = 0200 = 2.0
     0.5 + 0.25           0080 t 0040 = 00C0 = 0.75
     3.0 - 1.0            0300 - 0100 = 0200 = 2.0
     5.0 - (-1.0)         0500 - FF00 = 0600 = 6.0
     127.9961 + 1.0       7FFF + 0100 = 80FF = -127.0039
     -127.9961 - 1.0      8001 - 0100 = 7F01 = 127.0039

(b)  1.0 * 1.0            0100 * 0100 = 00010000 = 1.0
     5.0 * 0.5            0500 * 0080 = 00028000 = 2.5
     -1.0 * 2.0           FF00 * 0200 = FFFE0000 = -2.0
     -2.0 * -3.0          FE00 * FD00 = 00060000 = 6.0
     64.0 * 2.0           4000 * 0200 = 00800000 = -128.0

(c)  1.0 / 2.0            01000000 / 00020000 = 00000080 = 0.5
     2.0 / 1.0            02000000 / 00010000 = 00000200 = 2.0
     -1.0 / 2.0           FF000000 / 00020000 = FFFFFF80 = -0.5
     -2.0 / -4.0          FE000000 / FFFC0000 = 00000080 = 0.5
```

Figure ***4-a) Because the numbers use two's complement notation, fixed-point addition and subtraction work just*** like ***Integer arithmetic. Notice that overflow occurs for results exceeding the valid± 128 range. b) Multiplying two fixed-point numbers produces a result with twice as*** *many* ***bits.*** **The** ***product term is containedin the middle two bytes of the four-byte result. c) Dividing two fixed-point numbers requires two alignments. The Furnace Firmware code moves the divisor to the middle two bytes of a four-byte variable and puts the dividend in the high bytes ofanother four-byte variable.*** **The** result ***appears in the lower two bytes of the quotient.***

SUMMING UP

At this point, you have nearly everything you need to build an 8031 system that'11 blow their socks off: interrupt-driven ring-buffered serial I/O, console support for both LCD and keypad hardware, fixed-point math, nonvolatile RAM, analog inputs, timekeeping, and a great glob of glue code. All you need is the program that connects the pieces.

But do you need C? No, it turns out that most of the code doesn't depend on C at all. True, the calling conventions are peculiar to Avocet C, but those can be readily adapted to other compilers-or to your own assembly language standards, as you see fit.

Of late, I've ignored the BASIC-52 interpreter because shoehoming the **code into** that straitjacket requires too much fiddling around for my purposes here. However, I did write the LCD and keypad code with BASIC-52 in mind, so it would not be too difficult to support BASIC console I/O. I'll leave this as an exercise, and if you drop a note on the BBS after you've studied the problem for a while, we'll even help you out!

I'm not entirely satisfied with C for microcontrollers. The amount of code per line seems excessive, the C-language library routines are entirely too bulky, and there are, regrettably,

```
;--- conversion table entries contain
; 0     binary fraction bit location
; 1-2   packed BCD equivalant of that binary bit
; 3-4   packed BCD 9's complement of that bit
; must be ordered from highest to lowest bit for string conv.

FixEntry     %MACRO   b,d,n
             DB       b
             DW       d
             DW       n
             %ENDM

             SEG      Constants
             ORG      CONSTBASE+FXP_CONST

FixTable     FixEntry $80,$5000,$5000
             FixEntry $40,$2500,$7500
             FixEntry $20,$1250,$8750
             FixEntry $10,$0625,$9375
             FixEntry $08,$0313,$9687
             FixEntry $04,$0156,$9844
             FixEntry $02,$0078,$9922
             FixEntry $01,$0039,$9961
BITLOC       EQU      0                  ; offset of bit ID
BCDLOC       EQU      1                  ; offset of BCD
NEGLOC       EQU      3
ENTRYSIZE    EQU      5                  ; size of each entry
NUMENTRIES   EQU      8                  ; num of bits to process

;--------------
; Convert a fixed-point fraction into ASCII
;  char *FxpCvtFract(FIXPTNUM FPVal,char *String);
; Target string pointer is on stack
; Integer  part is in R4 (which we ignore)
; Fraction part is in R5
; Can't have overflow from fraction to integer...
```

(continued)

Listing 2—*Converting a fraction to ASCII is ordinarily a tedious operation, but using tables provides a shortcut.*

some vicious bugs. Is it better than BASIC? Yes, indeed. Is it better than assembler? Yes, mostly. Is it optimum? Umm...certainly not!

C can perform reasonably well if you are willing to devote an inordinate amount of time to creating your own library routines in assembler and keep an eagle eye on your code space usage. If your project permits that level of attention, C is probably the right way to go: you can write more complex logic with less effort than in assembler, for sure. The catch is that you may run out of either space or time sooner than you expect.

RELEASE NOTES

The demo routines for this column exercise the ANSI driver and fixed-point math packages. The other files include the additional functions needed for the fixed-point math code; most notable are `cgets()` (in `CONSOLEA.A51`) to handle input editing and strcat() (in `CLIBA.A51`) to combine two strings. The console I/O,

```
          SEG       UtilCode
_FxpCvtFract PROC
          PUBLIC    _FxpCvtFract

;--- convert fraction to packed BCD

          MOV       DPH,#HIGH FixTable ; point to table
          MOV       DPL,#LOW  FixTable
          MOV       R2,#0              ; clear BCD total
          MOV       R3,#0
          MOV       R1,#NUMENTRIES     ; number of loops
L?nextbit MOV       A,#BITLOC          ; pick a bit
          MOVC      A,@A+DPTR
          ANL       A,R5
          JZ        L?skipbit
          MOV       A,#BCDLOC          ; fetch BCD equivalent
          MOVC      A,@A+DPTR          ; high byte
          MOV       B,A
          MOV       A,#BCDLOC+1        ; low byte
          MOVC      A,@A+DPTR
          ADD       A,R3               ; combine with total
          DA        A                  ; adjust total
          MOV       R3,A
          MOV       A,B
          ADDC      A,R2
          DA        A                  ; adjust total
          MOV       R2,A
L?skipbit MOV       A,DPL              ; pnt to next table entry
          ADD       A,#ENTRYSIZE
          MOV       DPL,A
          DJNZ      R1,L?nextbit
;--- convert packed BCD to characters at pointer
          MOV       R4,2               ; move BCD to input loc
          MOV       R5,3
          CALL      _ConCvtHInt        ; pretend it's hex!
          RET
_FxpCvtFract ENDPROC
```

Listing 2—*continued*

LCD, and keypad routines were slightly modified to support the ANSI driver.

UP NEXT

Most benchmark programs measure execution time by repeating the same subroutine over and over again. That's OK if you only care about the average elapsed time, but what if you need to know how long a particular instruction takes? Get out your scope and find out, of course! I've wanted to measure 80x86 instruction timings for quite a while, and I think now's the time to start..✣

Ed Nisley is a member of the Circuit Cellar INK engineering staff and enjoys making gizmos do strange and wondrous things. He is, by turns, a beekeeper, bicyclist, Registered Professional Engineer, gunsmith, and amateur raconteur.

IRS

415 Very Useful
416 Moderately Useful
417 Not Useful

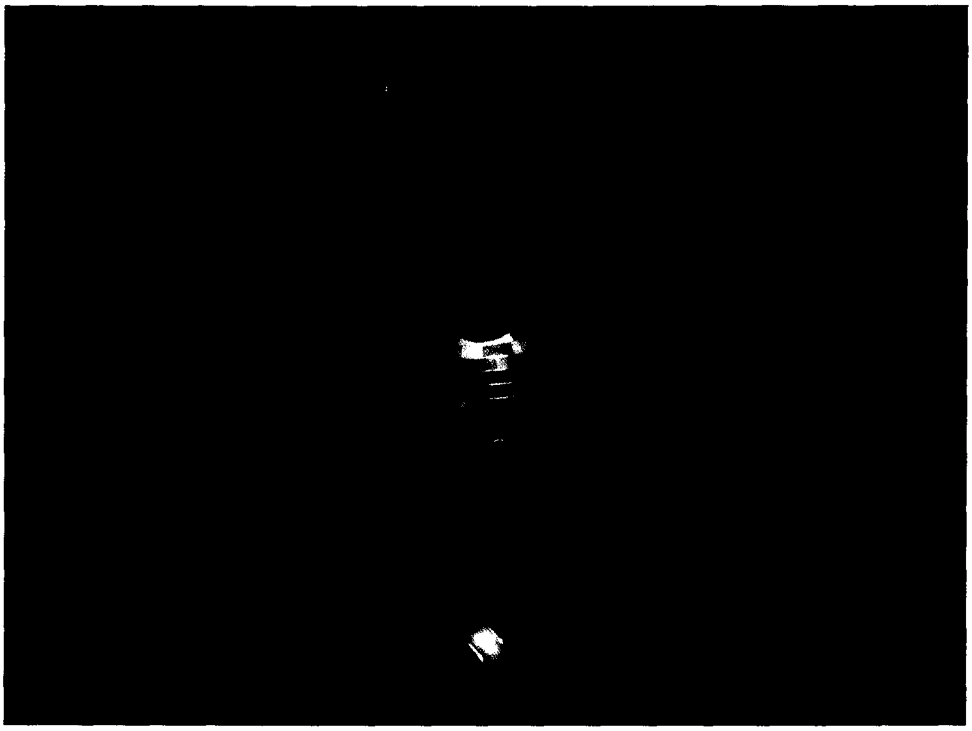
Magnetic Levitation: An Example in Closed-Loop Control

FROM THE BENCH

Jeff Bachiochi

How to Defy Gravity Without the Use of Black Magic

I've always had an urge to be a magician. While other kids were watching Mighty Mouse on Saturday mornings, I'd be tuned into Mark Wilson performing impossible feats of prestidigitation.



One of the more common illusions is the levitation of objects. The magician or illusionist will cause an object to literally rise or float without any visual means of support. Solid hoops or rings are passed around the object to confirm the reality of the situation. Even though we know there is logically some unseen means of support, we allow ourselves to be hoodwinked simply for entertainment purposes. Successful magicians are most proficient at disguising their secrets.

NO VISIBLE MEANS OF SUPPORT

If you've been in any gift/novelty store recently, you may have seen one of the "floating orb" novelties. This is a simple, freestanding frame in which an object, usually a sphere, is held visually unsupported in midair. Quite a stunning illusion!

Upon closer examination, you'll discover that magnetism is the key to this example of parlor trickery. A magnetic field is generated by an electromagnet hidden in the top of the frame and attracts the sphere upward. An infrared transmitter/receiver pair is mounted across the frame about one inch below the top. As the orb is gently lifted into the IR beam's path, the signal strength is reduced at the IR receiver. When the linear output of the IR receiver drops to about 2.5 volts, an electromagnet is switched on. As the IR reception is further reduced by the sphere rising toward the electromagnet, the voltage to the coil is also reduced. This reduces the magnetic field and gravity begins to win. When the sphere drops, the received IR increases which strengthens the magnetic field. An equilibrium is reached where magnetic attraction exactly counters gravity somewhere within the IR beam's path.

OPEN VERSUS CLOSED LOOP

Any control process can operate in an open- or closed-loop fashion. However, the term "open loop" is an oxymoron. As the word "loop" suggests, it is made up of a continuous path, whereas "open" suggests a noncontinuous path. Open-loop control is simply an action without any knowledge of the result, like turning on the porch light without looking out the window to see if it actually went on. On the other hand, closed-loop control adds some kind of verification that the control has had some effect on the task. One peek out the window confirms the result. Either the light did in fact go on, or it was on and you turned it off. Possibly, it is burned out or, even worse, another slip of paper gets added to your job jar. So, you can see that closed-loop feedback can not only show the process is operating correctly, but, just asimportantly, what has to be done to correct any errors in the output.

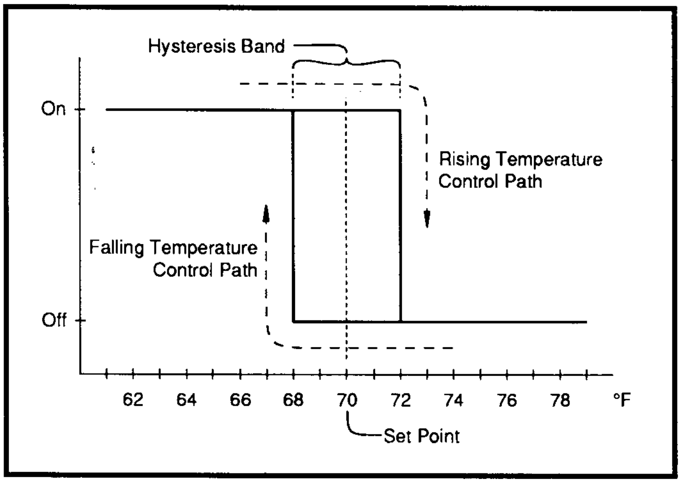


Figure 1—*Hysteresis is often used to prevent jitter on the control output.*

BALANCE OF POWER

Equilibrium can only take place when the strength of the magnetic field (the electromagnet) can overcome the force of gravity (the weight of the sphere) at the defined distance (IR beam to magnetic source). This defined distance must be less than the maximum distance at which the electromagnet can overcome the force of gravity.

The distance the sphere can travel, between the minimum and maximum IR beam interruption, is about 1 mm. The Law of Falling Bodies illustrates how long it will take the sphere to travel 1 mm:

$$S = \frac{1}{2} gt^2$$

where S is the distance in feet, g is the acceleration of gravity, and t is time in seconds. Rearranging and inserting numbers (with $S = 10^{-3}$ m and $g = 9.8\ m/s^2$), we get $t = 1.4 \times 10^{-2}$ s or t = 14 ms.

Worst case, if the control loop can't respond in less than 14 milliseconds, the sphere will drop beyond the range of magnetic attraction. Although process control brackets a wide variety of durations, most industrial processes are in the range of minutes, rather than fractions of a second as shown here.

CONTROLLER PRINCIPLES

Process control has three main parts: the process load measurement (error input), the controller (analog or digital), and the control element (correction output). The process load measurements can come from one or more sources, each having its own range about a nominal value. The control output can go to one or more control elements each having a maximum and minimum effect on the process. The controller must make decisions based on the process load measurements to adjust the control elements and return the process load to normal. Different algorithms may be used by the controller depending on the type of process being controlled. Each control mode has its own associated algorithm.

There are two main types of process control loops: discontinuous and continuous. The simplest is the discontinuous controller mode. Two-position control in this mode is simply on or off like the thermostat which controls the furnace in your house. Your thermostat measures the air temperature, makes a decision based on the temperature (and hysteresis) setting, and signals the furnace for more heat if necessary.

$$P = OFF \quad \text{if } E_p > t_s + h$$
$$P = ON \quad \text{if } E_p < t_s - h$$

where:

P = output to furnace
E_p = measured temperature
t_s = set temperature
h = half the hysteresis

Discontinuous mode may in fact have multiple set points. As each point is passed, the controller outputs a new level. In a three-position discontinuous mode, the output may take on a third intermediate value of, say, 50%, in addition to on or off. As the number of set points increase, you begin to approach a continuous controller.

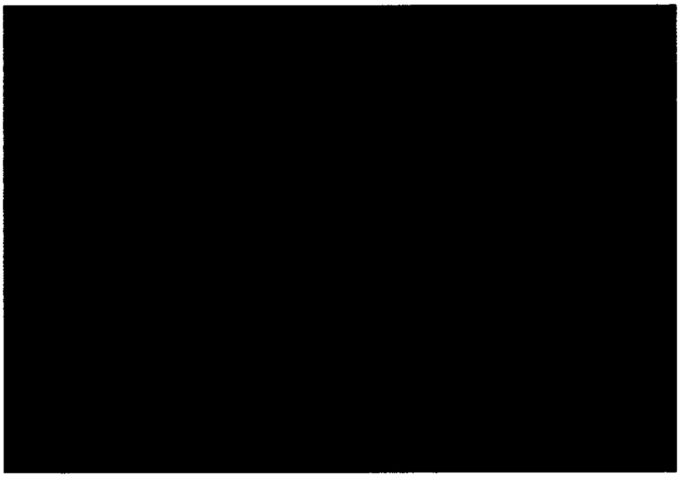


Photo 1—*The results of the hardware/software combination described later in this article clearly show the samples leading the input sine wave.*

There are typically three kinds of continuous modes: proportional, integral, and derivative. A proportional output has a direct linear relationship to the error input. It is used when system load changes are small because a permanent offset error occurs which limits the usable error span. A proportional output may be described as follows:

$$P = K_p E_p + P_0$$

where:

P = output
K_p = proportional constant (gain)
E_p = input measurement
P_0 = nominal set point

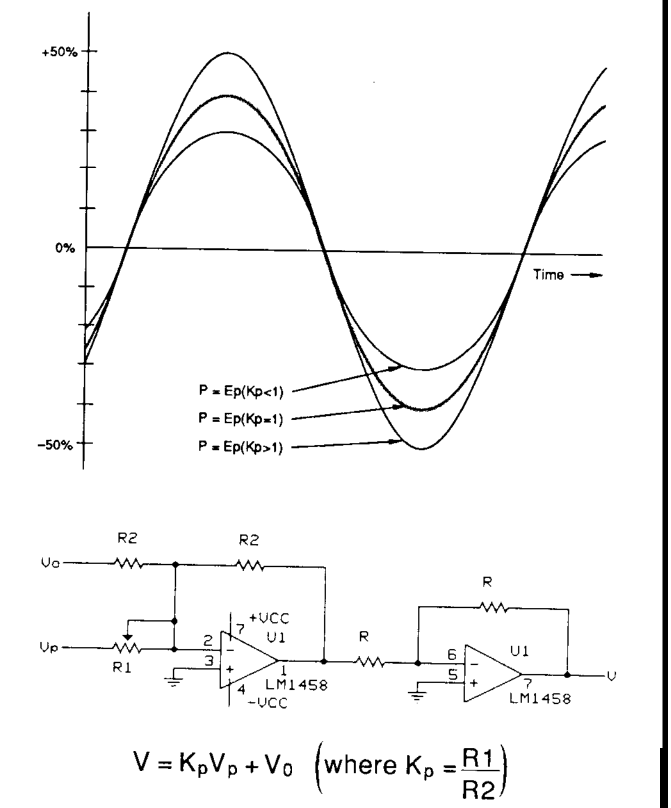


Figure 2—*The proportional control output has a direct linear relationship to the error input.*

An integral mode output changes faster for larger rates of error input, and drives the output at a slower rate as the error input approaches zero. This mode is normally used in combination with other modes where the process lag is short. Integral mode is defined by:

$$P(t) = K_i \int_0^t E_p(t)dt + P(0)$$

where:

$P(t)$ = output at time t

K_i = scaling constant

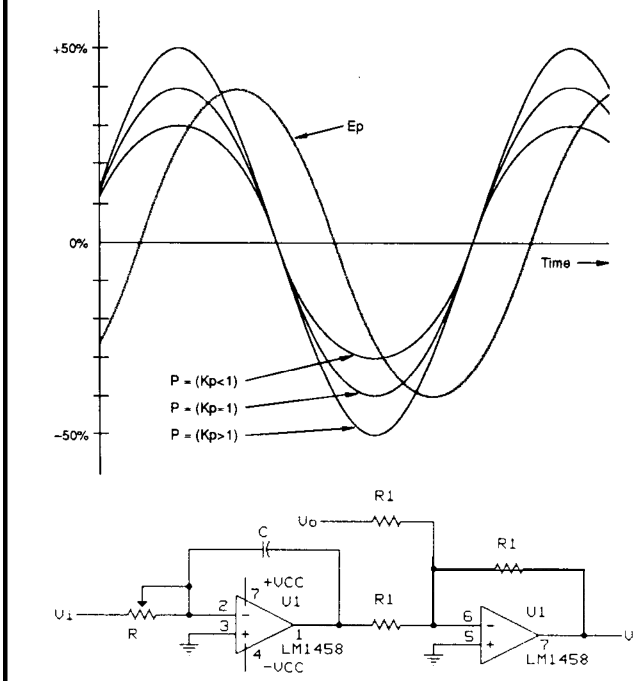
$\int_0^t E_p(t)dt$ = % change of input at time t from time 0

$P(0)$ = controller output at time zero

A derivative output is directly related to the rate of change of the error. That is, the output is driven not by the amount of error, but by how fast the error input is changing. Therefore, there is no output from either constant or no error, and maximum output from instant error. This mode is used in combination with the other modes due to its ineffectiveness when the input error is constant. The derivative equation is defined as:

$$P = K_p \frac{dE_p}{dt} + P_0$$

Reader Service #158



$$V = K_i \int V_i dt + V_0 \quad \left(\text{where } K_i = \frac{1}{RC}\right)$$

Figure 3-An integral *mode output changes faster for larger rates of error input and drives the output at a slower rate as the error input approaches zero.*

where:

P = output

K_p = gain constant

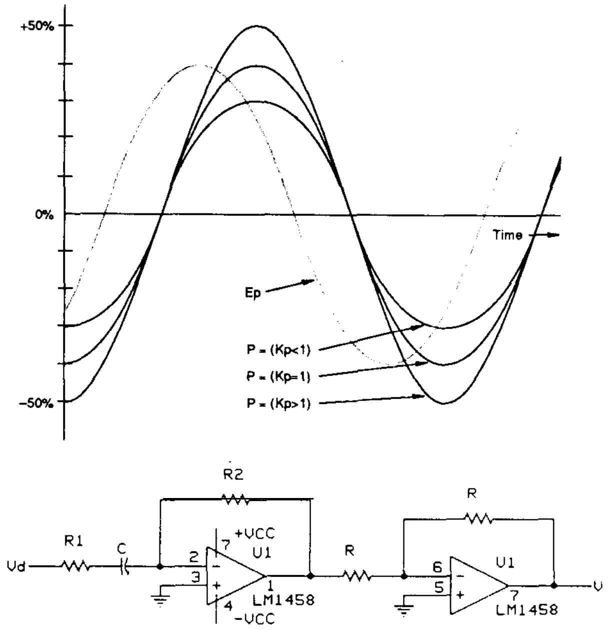
$\frac{dE_p}{dt}$ = rate of change of error

P_0 = output with no error rate

ADDING THE PROPER INGREDIENTS

Most industrial applications which use closed-loop control make use of a combination of controller modes. Thisisaccomplished by combining the proportional mode with either the integral or the derivative or both. The PI (proportional-integral) mode eliminates the offset error normally associated with the proportional mode. The I'D (proportional-derivative) mode is used to handle fast process load changes. While the PID mode can handle virtually all process conditions, it is the most complex.

Process control does not necessarily mean computer control. This is obvious in the simplicity of our home heating systems. Once other parameters enter into the picture, however, things change. Nightly temperature setbacks, different temperatures in each room, interfacing for external control (e.g., hand-held infrared) make control impossible for a simple mechanical thermostat. Microcontrollers give us the ability to change algorithms easily through software, which really beats getting out the old soldering iron and attacking the hardware to alter the



$$V = K_d \frac{dV_d}{dt} \text{ (where } K_d = R2 \times C)$$

Figure 4—*The output of a derivative mode controller is driven by how fast the error input is changing.*

offset, gain, integral, or derivative parameters every time we want to adjust the system.

APPLYING THE MAGIC OF MICROPROCESSOR CONTROL

Process control with a simple input and output doesn't require a microcontroller. However, using a simple process allows one to easily grasp the basics and the advantagesof using this approach on a more cost-effective basis.

With a small amount of signal processing, the position (error input) of the process load (ball) can be transferred to a microcontroller through an analog-to-digital converter, providing an error signal. The output control uses a digital-to-analog converter to vary the voltage of the electromagnet. The hardware selected for this project includes the RTC52 and RTCIO. *[Editor's Note: See "From the Bench" in the April/May '89 issue of* CIRCUIT CELLAR *INK (#8) for a description of the RTC52 and RTCIO boards.1* Both an A/D and a D/A converter are included on the RTCIO board.

Figure 5 shows the test setup which is fairly simple to construct. A hefty electromagnet's field is directed by a steel bar added to the top pole. This bar is bent down at each end creating flux lines which tend to pass through the sphere being levitated. An IR transmitter and receiver are placed below the electromagnet within the magnetic's limit of attraction. The analog circuitry to interface the IR input and solenoid driver is wired on a prototyping board.

The hardware stays the same no matter what algorithm software we use. Because the process load will be changing rapidly, I've chosen the combination propor-

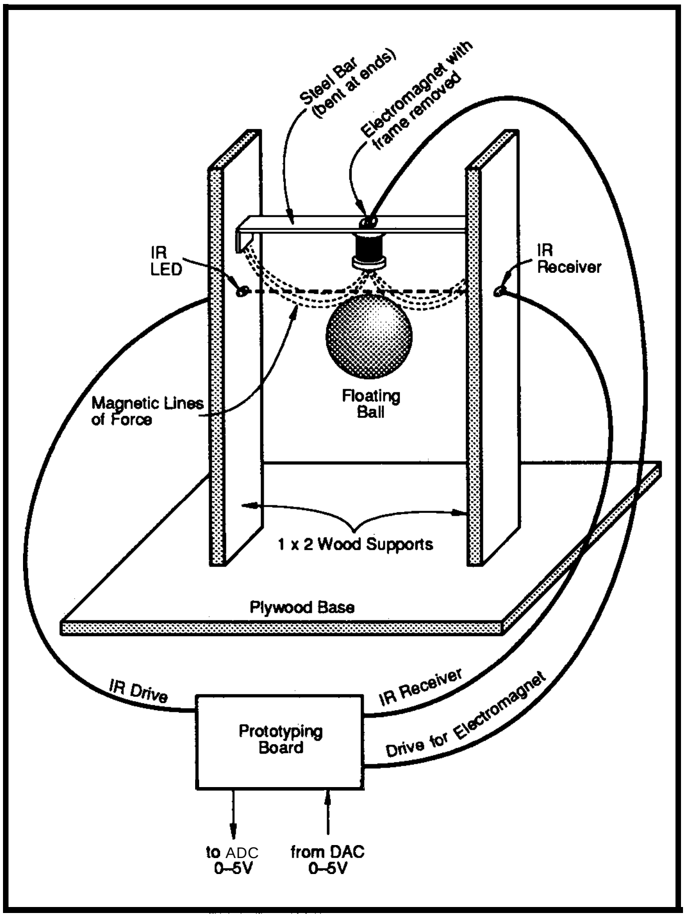


Figure 5— The test setup consists of an electromagnet, IR transmitter, IR receiver, support frame, and controller.

tional-derivative mode for this project. Here we have a process lag between the time the error is detected (by the IR beam) and the adjusted output voltage changes the magnetic field. This process lag can, if too long, cause the sphere to drop. If the lag didn't exist, the proportional mode would provide adequate control. Somewhere between the two extremes exists a cyclic condition where the ball merrily oscillates about the set point, gradually rising in amplitude between the control extremes. This generally leads to a loss of control. In this case, a bit of derivative mode control can make up for the process lag by adding additional error correction which is dependent on the error rate of change.

The equation for a combination proportional-derivative mode would be:

$$E = K_p E_p + \frac{K_d(E_p - E_0)}{tp - to} + P_0$$

where:

E = output
K_p = proportional gain
E_p = present input
K_d = derivative gain
E_0 = input at t_0
t_0 = time of last input
t_p = time of this input
P_0 = set point

Let's assume that:

$(t_p - t_0)$ = 0.001 s sample time
P_0 = 0 (no set point)
$K_p + K_d + P_0 = 1$ (no gain 5Vin/5Vout)

Therefore:

$$E = K_p E_p + \frac{(1-K_p)(E_p - E_0)}{0.001} + 0$$

where:

E = output
K_p = proportional gain %
E_p = input
$(1-K_p)$ = derivative gain (what's left after K_p)
$(E - E_0)$ = change in input error
O.&l = sample time (additional gain dependent on $[E_p - E_0]$)
0 = no P_0

Since the A/D and D/A converters both have 8-bit resolution, data in both directions will be in an unsigned character format of ranging O-255. If we use an integer in the range of O-10 to replace the normal gain value (%) and remove the factor of 10 later, all the math can be done in integer format, which speeds up overall program execution. The program shown in Listing 1 executes at 40 milliseconds per sample when run under interpreted BASIC-52.

At a 40-ms sample rate, the ball drops quickly to the floor. Fortunately, by compiling this BASIC-52 routine with BC151 Integer BASIC Compiler (also sold under the

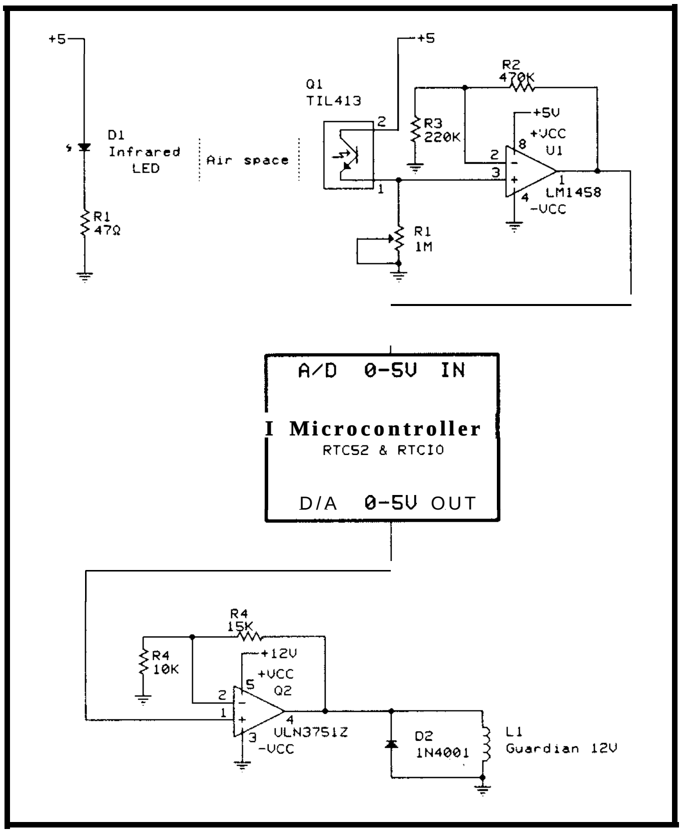


Figure 6-The interface for the 'floating orb' project shows the IR transmitter and receiver sections and the magnet control.

```
10 TCON=TCON.AND.0F7H : REM CLEAR INT1
20 TCON=TCON.OR.04H  : REM INT1=EDGE TRIGGERED
30 KP=9  : REM 90% PROPORTIONAL
40 EP=0  : REM INITIALIZE TO ZERO
50 EO=0 : REM INITIALIZE TO ZERO
60 KD=10-KP : REM 10% DERIVATIVE
70  V=((KP*EP)+(KD*(EP-EO)*100))/10: REM CALC
80 IF V<0 THEN V=0 : REM LIMITS MIN TO ZERO
90 IF V>255 THEN V=255 : REM LIMITS MAX TO 255
100 XBY(0E020H)=V: REM OUTPUTS VALUE TO DAC
110 XBY(0E010H)=0  : REM SELECTS CHAN 0 AND
      STARTS A/D CONV
120 IF(TCON.AND.08H)=00H THEN GOT0 120 : REM
      WAIT FOR EOC
130 TCON=TCON.AND.0F7H : REM RESET INT1 FLAG
140 EO=EP  : REM SAVE PRESENT SAMPLE AS LAST
150 EP=XBY(0E010H)  : REM GET A NEW SAMPLE
160 GOT0 70  : REM GO CALCULATE AGAIN
170 END
```

listing 1-*Software to control the orb consists of only a few lines of BASIC-52 code.*

name IBASIC-52 from Micromint), this routine runs at 1.4 ms per sample on an RTC52. This is a smidge over what I'd hoped for but still gives 10 samples during the 14 ms it takes the ball to fall 1 mm. This proved to be adequate.

Correction: In the RTC-V25 project presented in the last issue of CIRCUIT CELLAR INK, the master crystal should be labeled 16 MHz. A revised RTC-V25 schematic is available which contains this correction plus some design revisions. To obtain a copy, send a SASE to Circuit Cellar INK, 4 Park St., Vernon, CT 06066.

AND THERE YOU HAVE IT!

Photo 1 shows a slow sine wave fed into the A/D converter and the resultant processed D/A output. You can clearly see the samples leading the input. Adding the extra input and output circuitry to the microcontroller was the easy part. The software was twiddled slightly, decreasing theeffect in the derivative path, and the success proved exciting. Now if only I could make the ball disappear!

One last note: If you value any of your software, experiment with magnets at a considerable distance from your computer and media storage. Unless, of course, you wish to bulk erase everything in sight.

BIBLIOGRAPHY

"Process Control Instrumentation Technology" by Curtis D. Johnson (John Wiley & Sons).

Jeff Backiocki (pronounced "BAH-key-AH-key") is a member of the Circuit Cellar INK engineering staff. His background includes work in both the electronic engineering and manufacturing fields. In his spare time, Jeff enjoys his family, windsurfing, and pizza.

IRS

418 Very Useful
4 19 Moderately Useful
420 Not Useful

Goodbye CRT, Hello LCD

SILICON UPDATE

Tom Cantrell

I can't remember for sure, but I think my first experience with an LCD (Liquid Crystal Display) was during the digital watch craze of the mid-'70s. Everybody tried to get in the business-the boom attracted unlikely entrants including Intel where I worked (the idea was to sell chips with wrist bands). Needless to say, that episode isn't discussed in mixed company.

Anyway, my first digital watch had a seven-segment-type LED display. The only problem was that due to high powerconsumption,theLEDdisplaycouldn'tbeleftonall the time.Instead, the watch had a tiny button to turn on the display only when necessary.

Of course, the concept was an ergonomic nightmare. For instance, working the watch while talking on the phone meant either switching the phone into the shoulder/ear-jam position or executing a maneuver something akin to punching yourself in the nose. Checking the time while driving a car was also an exciting proposition, leading to the insight that it's better to be late for an appointment rather than never arriving (at least in one piece) at all.

Enter the LCD, a technology which, thanks to "reflecting" rather than "emitting" light, consumed so little power that the display could remain enabled at all times. Since it relied on ambient light, the LCD watches still needed a high-power lighting system (and the dam button) for use in the dark. In practice, there was usually enough daylight (or even moonlight) so that button-fumbling was kept to a minimum.

From humble wrist watch beginnings the LCD has slowly but surely improved and flourished. Now, LCD technology has reached the point that big changes in displays and associated products are on the horizon.

In fact, it can be argued that the historical mainstay display, the CRT, is threatened by the emerging LCD technology. Sure, it won't happen overnight, but read on and judge for yourself to what degree the LCD puts the CRT at risk of extinction.

THE GOOD, BAD, AND UGLY

As mentioned, the original and still claim-to-fame for the LCD is low power consumption. A brief explanation of LCD basics will show why.

The liquid crystal phenomenon actually was discovered over 100 years ago but the concept laid dormant until the '60s, when revitalized research finally gave us our '70s LCD watches and calculators.

Figure 1 shows the organization of a simple TN (Twisted Nematic) "reflective" LCD. Naturally, the key for LCDs is the "liquid crystal" material-a proteingoo. Likeacrystal, the material has the property of "turning" light in a certain direction. Like a liquid, the orientation of the crystal is "fluid"; in particular, the orientation can be changed by theapplicationofanelectronicfield.

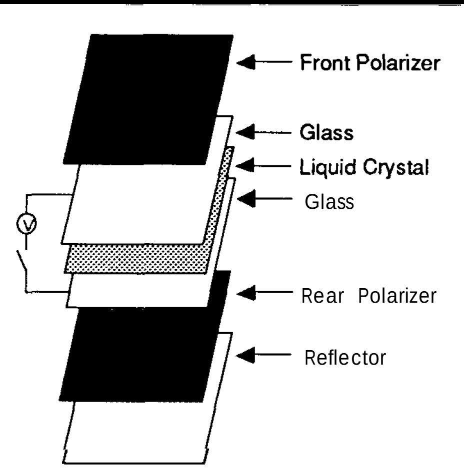


Figure 1 -The *organization of a simple TN reflective LCD.*

To make a display, the liquid crystal is sandwiched between a grid of electrodesand sheetsof glass. The other key are two polarizers (front and back) which are oriented 90 degrees out of phase. Polarizers have the property of only passing light which is "aligned' with the polarizer. Perhaps you once tried this experiment during the heyday of "polarized" sunglasses. Take two lenses and put them on top of each other. Now, look through the pair as you twist them relative to each other. Sure enough, more or less light passes as the alignment between the two lenses is changed.

Finally, behind it all is a reflector panel. Remember, the simplest LCDs rely on ambient light.

So, light enters the first polarizer and is aligned in one direction. Next, the aligned light enters the liquid crystal and is either "twisted" or not depending on whether the crystal isenergized or not. Light that is twisted by the crystal can successfully pass through the second polarizer (which is twisted relative to the first polarizer) to thereflector and ultimately back to your eager eyes. On the other hand, light which doesn't get twisted by the crystal is blocked by the second polarizer and thus not reflected.

Low-power operation is a result of the fact that theliquid crystal requires little power to retain orientation and, like a CRT, need only be "refreshed" periodically. Using a row/column addressing scheme, only a fraction of the panel's "pixels" (as few as **1/100** or **1/200)** are energized at any point in time. Thus, the basic "good" of **LCDs** is that power consumption is **tiny—typically 0.025–1** W depending on panel size. Contrast this to the 75 W or so for a typical CRT.

The only intrinsic 'bad" for **LCDs** is that the liquid crystal, like any other liquid, acts up when exposed to temperature extremes. Most panels are specified to operate in a range of **0–50°C,** but some are limited to even a narrower range (e.g., **10–40°C).** For thenonmetric out there, suffice it to say you shouldn't expect your LCD widget to work in a snowstorm or a heat wave.

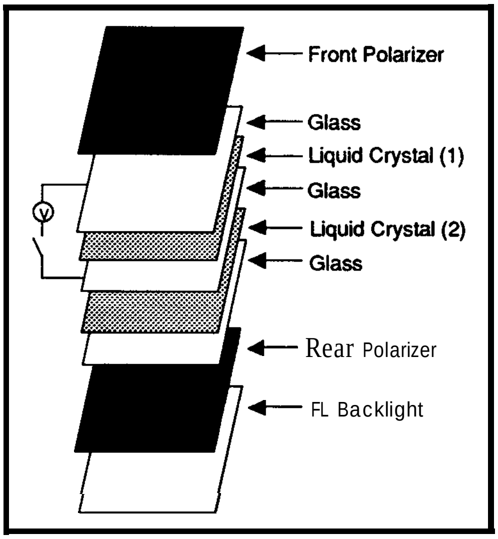


Figure **2—***The* **double** *super twisted nematic* ***display adds another layer*** **of** ***liquid crystal.***

The basic 'TN reflective" LCD described so far (the typical calculator/watch-type display) does suffer from a variety of "ugly" problems. Most basic is the reliance on ambient light. The solution is "backlighting." A variety of backlight technologiesareavailable which offer various tradeoffs for light level and color, size, durability, and power consumption. The major contenders are EL (electroluminescent) and FL (fluorescent) lighting schemes. EL **is** best for **low-**end applications due to reduced thickness, weight, and power consumption while, at the cost of worsening these three factors, FL generates a brighter full-spectrum light.

Another pair of related problems is limited contrast and viewing angle; both are byproducts of losses and variabilities in the light's twists and turns. Basic TN **LCDs** may have contrast ratios as low as **4:1** (read "light black characters on a dark gray background"-in other words, don't lose your **eye doctor's phone number). Furthermore, the useful view-**ing angle may be as little as 30-35 degrees. If you've ever tried to share an LCD display with another person (showing them something on the screen for example), you know it only works if both of you are real good friends. Unless you butt heads, one of you will see mush!

The solutions that **have emerged** for contrast/viewing angle problems are variations of the "Super Twist" theme.

The simplest is called STN (Super Twisted Nematic). The only difference from TN is that STN twists the light more which results in a 20–50% improvement in contrast and viewing angle.

The next iteration, DSTN (Double Super Twisted Nematic, Figure 2) is nothing more than the brute-force addition of another layer of liquid crystal. Contrast and viewing angle are dramatically improved and a true "black" is possible. The downside is, besides adding weight and thickness, DSTN needs brighter backlighting to overcome losses through the extra crystal/glass layer.

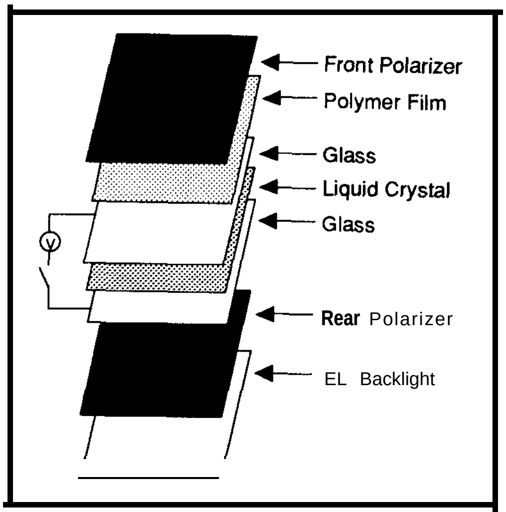


Figure 3—*The Film STN display adds a thin polymer film to fill the role of DSTN's second LCD.*

The latest version, Film STN (Figure 3) reverts to the STN single-layer format, but adds a thin polymer film to fill the role of DSTN's second LCD. Though contrast ratio for Film STN (about the same as regular STN, i.e., 10–12:1) isn't as good as DSTN's 18:1, the viewing angle is a much more usable 60 degrees or so.

Users of modern portable PCs (which wouldn't even exist without the LCD) can testify that the display quality is quite good-a far cry from the first generation of "squintable" laptops. Nevertheless, relative to the CRT, these LCDs suffer from three major weaknesses.

Most obvious is the lack of color: Generally, all the previously described LCDs are monochrome. Second, even the best LCDs don't achieve the contrast ratio of a CRT (one reason color isn't a good match). Finally, a flaw you can't see in a brochure's static screen shot is that the LCD display update rate is relatively slow. Any user of conventional LCDs knows that when the beautiful screen starts scrolling, it quickly turns into an unrecognizable blur.

Unfortunately, these limitations are in fundamental conflict with the emerging computer standards, notably the shift to Macintosh-/Windows-like color bit-mapped pointing-device-oriented visual interfaces. Even if monochrome is deemed acceptable, the slow-update problem is a real show-stopper.

TFT TO THE RESCUE

Now, a new LCD technology is emerging: TFT (Thin Film Transistor), also called "Active Matrix." As the names imply, the concept relies on placing a transistor at each pixel location in contrast to older LCD's remotely driven scheme.

The basic merit of the TFT approach is, thanks to reduction in cross talk, precise threshold control, and use of "less twisted" (i.e., faster) liquid crystal, the pixel can be switched very quickly by the local transistor. Plus, exploiting FL backlight, contrast ratio is dramatically improved compared to the older LCDs (e.g., 40:1 vs. 18:1). Figure 4 compares some general specs for the older LCD types (TN, STN, DSTN, Film STN) and the TFT LCD.

	TN	STN	DSTN	Film STN	TFT
Contrast Ratio	4–8:1	6-10:1	18:1	12:1	40:1
Response Time	200 ms	250 ms	250 ms	250 ms	40 ms
Power Consumption (with backlight)	0.13 w	0.3–7 w	7 w	3 w	6-15 W
Viewing Angle	35 deg.	40 deg.	45 deg.	60 deg.	>60 deg.
Thickness (with backlight)	5-12 mm	7-28 mm	28 mm	7 mm	30 mm
Weight	25-150 g	150-1800 g	1600 g	800 g	1090 g
Resolution	Low	Med	High	Med	Med
Color	Mono	Mono	Mono/Color	Mono	Mono/Color
Cost	0.5x	0.7x	1x	Ix	2.5x

Figure 4-A *summary of* LCD *technologies shows the benefits and drawbacks of each.*

The TFT LCD has a couple of fundamental advantages over our beloved tubes. The latter rely on phosphors to emit light while the LCD uses the combination of backlight and passive dyes. Thus, the LCD can achieve wider color range since the variety of dyes is much greater than that of usable phosphors (and remember, TFT LCD 'black" is really black). Also, since the CRT's phosphors emit light from the surface of the screen, dispersion in all directions introduces a "bleeding" effect. By contrast, in direct line of sight, the LCD shows absolutely no overlap between adjacent pixels.

The bottom line is TFT LCDs work real well now and they will only get better. In fact, the major barrier to widespread use isn't performance; I imagine 90% of PC users would be glad to junk their CRTs in favor of a TFT LCD today. The problem is cost, and a look at how TFT LCDs are made will shed some light on the million-dollar question, namely "why do they cost a million dollars."

THE WORLDS BIGGEST DRAM

Let's take a look at a state-of-the-art TFT LCD: the Hitachi TM26D01VC. The TFT LCD is a cross between a regular LCD and a DRAM as shown in Figure 5. Like a regular LCD, the TFT LCD relies on a sandwich of (from front to back) polarizer, glass, liquid crystal, glass, polarizer, and backlight. The only difference is that additional layers are placed on either side of the liquid crystal, in front a color filter panel and behind a thin-film of transistors.

Conceptually, the TFT LCD is simple. Each transistor is responsible for energizing the liquid crystal associated with the individual color components of a pixel. The Hitachi unit uses a "vertical stripe" arrangement in which each "pixel" corresponds to three (R, G, B) vertically aligned "dots" (the total pixel size, 0.33 mm, is about the same as a CRT). Thus, the 640 x 480 "pixel" panel is really composed of 640 x 480 x 3, or 921,600, transistors--about the same as a l-megabit DRAM.

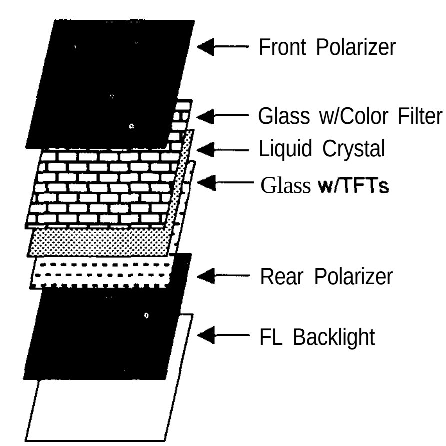


Figure **5—*The TFT* LCD is *a cross between a regular LCD and a DRAM.***

Indeed, the transistor setup is quite like a DRAM, with TFTs on glass substituting for the DRAM's MOSFETs on silicon. Unfortunately, while I can toddle down to local hacker emporium and pick up l-megabit DRAMs for little more than $5, the TM26D01VC is a much more serious proposition-"Hmm, should I get a color TFT LCD-based PC or a new car?"

You see, the cost has little to do with the number of transistors; after all, the fundamental force driving the IC revolution is more transistors for less money. Instead, it's the "die size" that kills yield. That's because the statistical chance of encountering a process/material defect grows much faster than the area itself. A die with ten times the area per transistor will cost much more than ten dies with one times the area per transistor.

Remember, little stands in the way of making silicon dies smaller-indeed, the problem these days is coming up with packaging/wiring schemes to handle ever more connections with an ever tinier die. Of course, the cost-reducing "die shrink" concept can't be applied to displays; shrinking a mainframe to fit on the head of a pin makes sense, but people like their screens a little larger.

Fabricating the TFT LCD panel is only part (though the most expensive part) of the story. The TM26D01VC also includes a number of control and driver ICs that together offer a digital video CRT-type interface quite similar to that of typical PC. The electrical interface between the "glass" panel and "silicon" chip portions of the panel is interesting: A "conductive organic adhesive" is used, bringing new meaning to the term "glue logic."

The backlight also plays a big role since stringent brightness and spectral characteristics are required for best results. Hitachi offers a unit (theTB2602) consisting of six fluorescent lamps and a diffuser. Note that while the

LCD itself consumes about 1 W (5 V @ 60 mA, 25 V @ 30 mA max) the backlight requires 17 W (12 V @ 1.4 A) to operate (36 W, 12 V @ 3 A, during start-up).

Unfortunately for our wallets, the TM26D01VC's 211-mm x 158-mm "die" is easily one-hundred times the size of even the latest and greatest "wunderchip." The term 'WSI" (Wafer Scale Integration) has been used to refer to forthcoming silicon "wafer" size (6"–8" diameter) chips. Since the TFT uses a similar-sized pane of glass, "WSI" could also mean 'Window Scale Integration." Either way, "WSI" means big bucks and, for now, tight availability.

According to Hitachi, most OEMs can get a panel or two to play with at about $3500 a shot. Though the price is lower, say $2000, for manufacturing quantities, it's kind of moot since production is just getting up to speed. I imagine it will be two years before panels are widely available.

SAFETY FIRST

I predict that another factor will emerge in the LCD's favor: Safety! In my opinion, it is only a matter of time before CRTs are plastered with warning labels alerting the user to the potential deleterious effects of electron bombardment and/or low-frequency EMF.

Note that I am certainly not qualified to say whether or not a safety "problem" really exists with the CRT. However, in the absence of hard evidence, the 'belief" that a safety problem exists will be enough to indict the CRT.

Make no mistake, the issue of CRT safety has the potential of becoming a big economic-political-legal-medical-technical-emotional battle unless studies can conclusively prove it's not a problem.

Of course, you've probably guessed by now that the LCD doesn't have the CRT "emission" and "radiation" problems (although I'm not sure about the backlight). Assuming performance and price advantages of the CRT are slowly but surely eroded by the TFT LCD, isn't it only a matter of time before the "safety" issue comes to the fore?

OK, now you can go move your CRT back a tad..

Contact

Hitachi America, Ltd.
Electron Tube Division
300 N. Martingale Road, Suite 600
Schaumburg, IL 60173
(708) 517-1144

Tom Cantrell holds a B.A. in economics and an M.B.A. from UCLA. He owns and operates Microfuture, Inc., and has been in Silicon Valley for ten years involved in chip, board, and system design and marketing.

IRS

421 Very Useful
422 Moderately Useful
423 Not Useful

CONNEC-TIME

Conducted by Ken Davidson

Excerpts from the Circuit Cellar BBS

The Circuit Cellar BBS
300/1200/2400 bps
24 hours/7 days a week
(203) 871-1988
Four Incoming Lines
Vernon, Connecticut

These are busy times on the Circuit Cellar BBS. In this issue, we'll be covering discussions about state machines, modem usage in other countries, and a nifty TV antenna now on the market. Let's start off with a subject near and dear to those just starting ouf in engineering: school.

Msg#:31195
From: ROBERTO PUON To: ALL EEs

I've had electronics as a hobby for around four years. Most of what I know I have learned either by trial and error or from The Circuit Cellar. Right now I am a junior in college and I am an EE major. I've not taken but one (digital) electronics course. I really enjoy electronics, and if my job is going to be as fun as my hobby has been, work is going to be a party!

I have one question for all of you Electrical Engineers which you might have asked yourselves when you were in college: How much of the math that you studied in college do really use in the field?

Msg#:31211
From: JAMES D STEWART To: ROBERTO PUON

As a non-degreed engineer, who worked his way up from fixing missiles in the Army in 1970 to project leader presently, I would guess very little. Ohm's law and basic math skills are all I've ever used. You could probably argue that they are all that I have. The one place that I have been stuck in the past is trying to understand algorithms used with digital signal processors. It hasn't been a serious problem yet, and my contingency plan is to hire a tutor as needed when I start to work with them seriously. The University of California at Berkeley is next door and there are lots of math giants willing to work cheap.

I think the amount of math that you need is greatly dependent upon the type of engineering that you are doing. Most engineers doing microprocessor development do very little math. If you work in communications, process control, or signal processing, you will need more higher math. I feel that if I had had more math, perhaps I would be better able to visualize problems and find quicker solutions.

Msg#:31279
From: KEN DAVIDSON To: ROBERTO PUON

That sums it up pretty well. When I was going through school, I kept saying, "But that's not the way it works in the *real* world." Strictly speaking, with digital electronics you rarely worry about poles and zeros and Norton equivalents, but it's absolutely necessary (in my opinion) to have a solid background in that stuff just so you know why things work the way they do. Much of the theory and math doesn't directly apply to real-world engineering, but you often have to keep it mind when deciding on design tolerances and what you can and can't get away with.

I have a friend that I graduated with who is working on cordless phones at AT&T. His digital experience is pretty weak, but he's an ace analog designer. He uses a lot of the math and theory every day that I've been exposed to in the past, but was happy to put behind me. I was just talking with him this weekend and he was saying something I agree with: you don't go to engineering school to learn about what chips are out there or the best way to lay out a PC board. You go to school to learn the basics of the design process and to learn how to learn.

The best thing that can happen to you when you graduate is to be paired with a 20-year veteran. He'll be able to teach you things you never dreamed of in school, but with the proper background, you'll be prepared to understand why things work the way they do and how to do things the "right" way. There is absolutely no way you'll be able to come straight out of school and design a circuit that works, is cost effective, and is easy to manufacture (unless you've had hobby or summer experience, but even then you'll probably need some help). Hell, we weren't even taught how to solder while in school.

Just keep plowing through it and telling yourself you'll be a better person for doing it. ;-)

Msg#:31280
From: STEVE CIARCIA To: ROBERTO PUON

I are an engineer too. :-)

How much math you use is really dependent upon the job. If you are assigned to design radar antennas and don't know math you are in big trouble and soon out of that job. Just using off-the-shelf ICs in noncritical control designs, with the manufacturer's specs under one elbow, is not particularly math intensive.

In my opinion, you cannot be a good engineer without knowing a good deal of math. This math could be broad ranging, including physics and chemistry, and not necessarily just narrow hard-headed stuff for designing your own ICs. A good engineer (EE) will know what Fourier transforms are, but 20 years after studying them he might not know how to solve them by hand. A good EE will instead know immediately where to find a book on the subject, a math egg-head to do it for him, or a computer program that will do it. If all else fails, he can hope that he's been on the job long enough to have enough seniority to reassign the task to the new kid on the block and go back to his tough management job (managing the coffee and doughnut fund).

Engineering is 90% intelligent gut reaction and 10% mathematical analysis. The only problem you've got as a young engineer is that the majority of old geezers on the project will be winging it by then on politics and reputation and the new guy with have to do the work. In that case you better know some math.

Don't get me started on this. If you are in school, then learn something. If you really shouldn't be an engineer, don't find it out on the job.

Msg#:31295
From: ED NISLEY To: ROBERTO PUON

The single most fundamental mathematical operation you'll ever learn is long division...and I was taught this long after I graduated, so you can save some time.

What you do when you're starting work on a problem is take two critical parameters and divide one into the other. If the ratio doesn't make sense, the project is usually in serious trouble. When you do this sort of thing in your head you develop a reputation for Great Wizardry.

Unfortunately, in order to know which two numbers to divide and when the result makes sense requires a solid background in the math you'll get in college. For example, you need to know how fast is do-able, how strong is something about "that thick," and how heavy is something about "that big"... which you get from knowing physical and electrical properties.

I took the dreaded Fields & Waves courses in the Physics rather than the EE department because I really gummed up my schedule (took a semester off killing calves at a medical research center, but that's another story). The EEs learned about transmission lines, klystrons, and stuff like that. In the physics classes, we did the really grim math (I knew I was in good company when the Physics major in the back row asked 'What's a tensor?" after the first lecture...) the hard way. But, while I can't do any of that stuff any more, I have a reasonable appreciation for why you build high-frequency stuff the way you do.

No, you'll not need much of what you learn, but everybody uses a different portion. The value of your education is what you know when you've forgotten everything you learned in class... think about it.

All too often, we get stuck in the mindset that for a digital circuit to do any useful work, a processor must be included. Many times, though, a simple state machine may do the trick using less board space and at a much lower cost (in terms of both actual hardware and design time).

Msg#:31396
From: DAVID M. WILLIAMS, SR. To: ALL USERS

Does anyone have or know where I can obtain plans for a digital cypherlock. I have a twelve-key keypad and need something for the kids to use to activate the garage door opener. I had a unit made by 3M but a lightning strike took it out and it's to hard to get parts for the board. Seems 3M had some chips tailored made and no one including 3M or Harris, who made the chips, has replacements. If you know of some plans, I would appreciate some info.

Msg#:31566
From: BOB PADDOCK To: DAVID M. WILLIAMS, SR.

Well, if your keypad is a 2-of-7 or 2-of-8 type and not the multiplexed type (I was just about to send a message to Ed, about the overlooked virtues of the 2-of-*x* keyboards) you can make a simple electronic lock.
What you need is a 2-of-x keyboard, a 27(C)512 EPROM, a 74(HC)273 8-bit edge-clocked latch, and a 74(HC)14 which is used for reset and a slow oscillator. Feed the outputs of the EPROM into the inputs of the 273, feed the outputs of the 273 into AO-A7 of the EPROM. Connect your keyboard to A8–A15. Use the 14 to make an oscillator and reset for the 273. Connect D7 from the EPROM to one of the 14's inverters, and use that to drive a 2N2222 or equivalent that will in turn drive a relay.

What you have just built is a state machine, where the 273 represents the current state, with the keypad supplying the next state information.

What you put in the EPROM will determine what the thing does. For example, with 256 states you could have use a 128-digit number to turn the output on, with 128digit number to turn the output off (a bit unrealistic for us grown-ups to remember, but the kids can handle it :-)). Another way to program it is for 32 4-digit "on" commands, with 32 4-digit "off" commands, and so on. Just depends on how you want it to work.

These EPROM state machines are the thing to use where cost is an overriding factor in a design; at least until the Creeping-Features-Creature shows its ugly head and you need the flexibility of a CPU.

Msg#:31577
From: DAVID M. WILLIAMS, SR. To: BOB PADDOCK

Thanks for the suggestion. I just got an EPROM burner last month. I'll look this over and see if I can put one together. My pad is the 3 x 4 type, pressure film that supplies a ground (or power) through the key.

A prerequsite for any kind **of** *telephone data communications is,* **of** *course, a modem. What happens when you want to bring it along when traveling, though?*

Msg#:31603
From: BOB PADDOCK To: ALL USERS

One of my customers is on his way to the United Kingdom, with a Hayes 2400-bps compatible modem (internal modem for a Toshiba 5200); he wants to be able to call me with it when he gets there.

Has anyone here had any experience with making and receiving international modem calls? What pitfalls should we be looking out for?

Msg#:31765
From: PELLERVO KASKINEN To: BOB PADDOCK

The main difficulties in international modeming seem to be covered, as the modem is 2400-bps type and possibly even supports thedialingin the European network, at least in the pulse dialing mode. And in any case, the dialing can be handled manually if the modem does not happen to comply. What remains is the physical connection. If I am right, the British system is something else than modular jack based. In Ireland, I understand, the modular jack has been introduced by Northern Telecorn, but that is a different story. So, have some good alligator clips included in the tool kit!

There are also the regulatory issues. In most of Europe, the telephone system is a monopoly and, like monopolies, they try to promote their own business. In this case it may mean that a renting of the modem from the telephone company is required, at least officially. Let's just hope that Margaret Thatcher has obsoleted those kinds of requirements since my last visit to that country!

Msg#:32142
From: BOB JENNER To: BOB PADDOCK

I don't have personal experience, but I've come close enough to ask the question. The answers I received suggested that the things to watch for are different modem signal protocols, the noisy and long-turnaround phone lines, and the rather different political setup: using someone's phone line for an unauthorized purpose, like data communications, can jeopardize their ability to have that phone line. At least one person suggested using a service that goes overseas rather than making the long haul directly.

Good luck and good traveling!

I'm sure you've seen the print ads for those silly TV-top "satellite dish antennas." Here's one caller who wanted **to** *check their validity (the original poster's name has been changed to spare him any embarrassment).*

Msg#:31442
From: GEORGE MASON To: ALL USERS

Does anyone know of a source that I can look up info on TV antennas? A friend of mine is living in an area where there are no cable TV lines run. Recent small talk has brought up a rumor that a company has developed an antenna which sets inside your window and is sort of like a mini satellite dish. Ever heard of this? This dish sounds more appealing than a huge roof-top antenna. Any help would be appreciated.

Msg#:31450
From: STEVE SAMPSON To: RICHARD MITCHELL

I've seen the ads. If it's the same one I saw I never had such a good laugh in my life! The thing you need to know is that VHF and UHF dish antennas are not as effective as beams. The little dish that you set in the window is for decoration only. It works no better than a set of rabbit ears. If rabbit ears will suffice, and you want to look like a yuppie with a mini-dish, then by all means. Otherwise, you can go two routes: buy a beam antenna cut to the exact frequency wanted, or buy some wide-band beams such as those sold by Radio Shack. Depending on the distance and terrain, nothing may help. I wouldn't plan on reception over a hundred miles. Where do you get these things? In big cities. There's usually an antenna specialist company. I don't have a good answer there. UHF dish antennas are becoming popular but these area couple of feet wide, and the "programming" is not worth the reception.

Msg#:31530
From: ED NISLEY To: STEVE SAMPSON

My buddy Dave Long (who had 37 antique TVs at the height of his craziness [and you should hear _that_ story]) built a 5-foot parabolic mesh dish and mounted it on (one) of his antenna masts. He's in Poughkeepsie and gets tolerable reception from Philadelphia, 150 miles away as the crow flies. On the other hand, he picked his house by reading the Dutchess County topo maps, finding the highest points, then checking for a house with a basement big enough for a rifle range. Found one, too.. .

He also built (of mesh and fiberglass resin) a 6-meter power-steerable satellite dish back in the days when this was the cutting edge and LNAs were so expensive your eyes fell out. He mounted that one on footings that look like a civil engineering final exam; pity the next owner who really wants a lawn at that spot.

Msg#:30905
From: FRANCIS DUNLOP To: ALL USERS

I have written a keyboard interrupt handler that intercepts scan codes, reassigns some of them, and so on. It works fine except I don't know how to turn on/off the keyboard lights (Num Lock, Caps Lock, Scroll Lock). Is there some command that you send to the keyboard to do this? I have noticed that if you toggle bits at memory address (40:17) or thereabouts and then let DOS have control of the keyboard, the lights go on and off. However, for my real-time application I can't let DOS have control. Any help would be appreciated! Thanks.

Msg#:30920
From: DALE NASSAR To: FRANCIS DUNLOP

I don't know if it will do exactly what you need, but you can control the three keyboard lights (AT only) by issuing the following instruction: OUT `&H60,&HED` followed by OUT `&H60,&Hxx`, where xx is represents the lights in binary-bit 0 for "Scroll Lock," bit 1 for "Num Lock," and bit 2 for "Caps Lock." For example, `&H5` turns on Scroll Lock and Caps Lock and turns off Num Lock. Keep in mind that changing the lights in this manner does not change the actual shift states which are controlled by bits 4, 5, and 6 at address 0040:0017 (BIOS KB_FLAG)

Msg#:31055
From: ED NISLEY To: FRANCIS DUNLOP

Dale's got the direct keyboard interface knocked, but you can also diddle the shift flags and call the keyboard BIOS (INT 16h, AH=01h) to have it do the updating. I don't know if your application can stand to have the BIOS in there, but that's surely the easiest way to do it. A little experimentation should tell you if the BIOS goes off into the bushes for longer than you can stand.

IRS

424 Very Useful
425 Moderately Useful
426 Not Useful

The Circuit Cellar BBS runs on a 10-MHz Micromint OEM-286 IBM PC/AT-compatible computer using the multiline version of The Bread Board 'System (TBBS 2.1M) and currently has four modems connected. We invite you to call and exchange ideas with other Circuit Cellar readers. It is available"24 hours a day and can be reached at (203) 871-1988. Set your modem for 8 data bits, 1 stop bit, and either 300, 1200, or 2400 bps.

SOFTWARE and BBS AVAILABLE on DISK

Software on Disk
Software for the articles in this issue of Circuit Cellar INK may be downloaded free of charge from the Circuit Cellar BBS. For those unable to download files, they are also available on one 360K, 5.25" IBM PC-format disk for only $12.

Circuit Cellar BBS on Disk
Every month, hundreds of information-filled messages are posted on the Circuit Cellar BBS by people from all walks of life. For those who can't log on as often as they'd like, the text of the public message areas is available on disk in two-month installments. Each installment comes on three 360K, 5.25" IBM PC-format disks and costs just $15. The installment for this issue of INK (December '90/January '91) includes all public messages posted during September and October, 1990.

To order either Software on Disk or Circuit Cellar BBS on Disk, send check or money order to:

Circuit Cellar INK — Software (or BBS) on Disk
P.O. Box 772, Vernon, CT 06066

or use your MasterCard or Visa and call (203) 875-2199. Be sure to specify the issue number of each disk you order.

STEVE'S OWN INK

Steve *Ciarcia*

The Whole Job

I finally did it. After several happy years using an 80286-based clone computer, I joined the 1990s and installed an 80386 fire-breather on my desk. I suppose that I should be feeling pretty excited about the whole affair, but the transition focused my attention on a subject that shouldn't come up any more. As I went through software installation on all of the various programs I've come to depend on, it became painfully obvious that some programs were guided by the hand of a marketing staff and some by the hand of an engineer. I have a definite preference for one type, and I think you'll be surprised to hear which one.. .

I'll start by describing two installation procedures. It doesn't really matter which program I'm talking about, because there are scores of packages that use each method. What does matter a great deal is the attitude each method exhibits toward the user.

In the first installation, I turn to the manual section titled "Getting Started." Following the directions and illustrations, I insert Disk #1 in drive A, type "Install." Menus, windows, animated palm trees, and pop-up instructions guide me through all 27 screens and 7 disks worth of installation. When I'm through, a new subdirectory tree has been created, and I can use the manual as a guide to check on the contents of my AUTOEXEC . BAT and CONFIG.SYS files.

In the second installation, I turn to the manual and wonder if a section has been left out. Finally, I find some installation instructions in Appendix 4.6.A. They say "Copy contents of program disks to hard disk. Set FILES= and HANDLES= for maximum performance." Oh boy. I copy all of the disks into a subdirectory I created, type the program name, hit ENTER, and...nothing. Checking the program disk, I see subdirectories. Inside the subdirectories are more subdirectories. Finally, I pull out a DOS shell program to automate the copying and, over an hour later I'm ready to start. I have to do some test runs to fine-tune my AUTOEXEC and CONFIG settings but, two hours after I start, I at last have a working program.

Here's my guess: The first program was developed with plenty of input from the marketing folks. They insisted on all of the pretty graphics (which waste precious CPU cycles) and they forced the programmers to sit with the tech writers for the manual construction. The second program, on the other hand, had a team leader with a BSCS or BSEE. They figured that anyone who used their software ought to know their way around a computer, so they decided not to "waste time" on making things pretty (and inefficient). Have you guessed which program I would recommend to other engineers? Just in case there's some doubt, let me clear things up-the marketing approach is the way to go.

Too many technical professionals feel that their job ends when the testing is finished. I'm convinced, though, that working hardware (or software) is only part of the job. Getting to working hardware or software may, in fact, be the easier part of the total. The rest of the job is making that product easy for a customer to use. Documentation, help files, user interface (with or without "graphical" in front), and packaging are all crucial parts of a good professional product. All too often, they're parts that are left to undertrained people struggling to make a deadline that the technical professional ignored. It's a shame that the aspects of a product that play the majority role in forming a customer's impression of the package receive the least attention from the engineers.

Writing isn't easy. Programming pop-up windows takes time that could be devoted to optimizing the last 10% of the program. Indexing a manual isn't nearly as much fun as designing a fast frame-grabber board. In spite of all these facts, the "human interface" is an important part of the job; too important to be left to people who didn't have a hand in the development of the hardware and software.

I'vealwaysbelieved that theengineer should write the manual, but the struggles with upgrading my computer made me go back and look at the manuals we've written. In many cases there is definite room for improvement. I'm an engineer, not a marketing guru, but I can learn a lesson: Engineering's just a hobby if there's no one to buy your product. There simply isn't a more important engineering function than building respect and care for your customer into the product.