CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

#76 NOVEMBER 1996

DIGITAL SIGNAL PROCESSING

High-Performance DSP

Real-Time DSP Modem

Empowered Digital Filtering

Self-Tuning PD Algorithm

11

0 74470 75349 0

$3.95 U.S.
$4.95 Canada

TASK MANAGER

Let Me Ask You a Dumb Question

I know this is probably a very stupid question, but have you looked at the price of PCs and memory these days? If you're perfectly content running your trusty '286 or '386 and haven't been paying attention, it's time to wake up.

I've been pricing systems during the past few days with the intention of upgrading some machines around the office. As I mentioned a couple months ago, we're going to standardize on Windows 95, so everyone needs at least a '486 with 8 MB of RAM. I started looking at cheap 4-MB '486s, but by the time I added 4 MB of memory, an 8-MB Pentium was already cheaper. Needless to say, we're getting a bunch of new Pentiums.

Similarly, my original estimate for the cost of memory to upgrade existing '486s was $20 per megabyte. When it came down to brass tacks and I started looking up real prices, it was suddenly down to $5 per megabyte. I almost had to look for excuses not to bring everybody up to 16 MB instead of "just" 8 MB.

As the success of our Embedded PC section shows, the use of PC-compatibles in dedicated applications is becoming more and more reasonable. With all the spare '286 and '386 systems being tossed aside for cheap upgrades and the wealth of software-development tools available for the PC environment, the size and the operating environment become the only obstacles to using cheap desktop motherboards and peripherals.

Two of our feature articles attest to this trend. In the first one, Matt Park and Brian McLeod explore the use of a newer PC sound card for doing real-time DSP modem processing. In the other, Dan Nygren relates a real-life story of how he recycled his desktop machine into a true embedded application.

Among our other features, Bob Fine introduces us to the ADSP2100 family of digital signal processors, Frank Gao looks at software algorithms for implementing a V.34 modem, and Brian Senese exposes us to the nitty-gritty of digital filtering.

Ken Baker also checks out a nifty technique for enabling a PD control system to tune itself to its environment, and David Rector wraps up our series on designing with EPLDs.

In this month's columns, Jeff covers the basics of schematic-capture packages, and Tom reports on the latest fire fight in the memory-card battles.

Last, but not least, come this year's Circuit Cellar Design Contest winners. Every year, it's a tough call as readers send in their best efforts and make our jobs harder. Congratulations go out to all the entrants. We already have a few articles in the works that more fully describe the projects, so keep an eye out for them in the coming year.

editor@circellar.com

CIRCUIT CELLAR®

THE COMPUTER APPLICATIONS JOURNAL

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

EDITOR-IN-CHIEF
Ken Davidson

MANAGING EDITOR
Janice Marinelli

TECHNICAL EDITOR
Elizabeth Laurençot

ENGINEERING STAFF
Jeff Bachiochi

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
Rick Lehrbaum
Fred Eady

NEW PRODUCTS EDITOR
Harv Weiner

ART DIRECTOR
Lisa Ferry

PRODUCTION STAFF
John Gorsky
James Soussounis

CONTRIBUTORS:
Jon Elson
Tim McDonough
Frank Kuechmann
Pellervo Kaskinen

PUBLISHER
Daniel Rodrigues

PUBLISHER'S ASSISTANT
Sue Hodge

CIRCULATION MANAGER
Rose Mansella

CIRCULATION ASSISTANT
Barbara Maleski

CIRCULATION CONSULTANT
John S. Treworgy

BUSINESS MANAGER
Jeannette Walters

ADVERTISING COORDINATOR
Dan Gorsky

CIRCUIT CELLAR INK®, THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (860) 875-2751. Periodical rates paid at Vernon, CT and additional offices. One-year (12 issues) subscription rate U.S.A. and possessions $21.95, Canada/Mexico $31.95, all other countries $49.95. All subscription orders payable in U.S. funds only, via international postal money order or check drawn on U.S. bank.
Direct subscription orders and subscription related questions to Circuit Cellar INK Subscriptions, P.O. Box 698, Holmes, PA 19043-9613 or call (800) 269-6301.
POSTMASTER: Please send address changes to Circuit Cellar INK, Circulation Dept., P.O. Box 698, Holmes, PA 19043-9613.

Cover photography by Barbara Swenson
PRINTED IN THE UNITED STATES

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

Circuit Cellar BBS—24 Hrs. 300/1200/2400/9600/14.4k bps, 8 bits, no parity, 1 stop bit, (860) 871-1988; 2400/9600 bps Courier HST, (860) 871-0549. World Wide Web: http://www.circellar.com/

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

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

INSIDE ISSUE 76

READER I/O

BILL COMES THROUGH

In my article on SDS software ("The Evaluation Board Saga Continues," *INK* 70}, I mentioned the consequences of sending SDS a fax asking for help in my microprocessor classes. I received some excellent demo software that I used in my class and in the development of a book. Since that fax produced so much, I concluded by musing about what would happen if I sent one to Microsoft.

Bob Davidson at Microsoft read the article and got the ball rolling. The marvelous result was a generous donation by Microsoft of NT server and client software, along with Visual Basic and Visual C++. Also included were memberships to Technet and Microsoft Developer Network and training materials. The total value of these materials at current *educational* discounts is $34,042 [U.S.).

We are very pleased to receive this donation. Many thanks to Microsoft and to Todd Needham, Manager of University Research Programs at Microsoft, for his help in providing this donation to the University of Calgary.

Mike Smith
smith@enel.ucalgary.ca

Contacting Circuit Cellar

We at Circuit Cellar Mencourage communication between our readers and our staff, so we have made every effort to make contacting us easy. We prefer electronic communications, but feel free to use any of the following:

Mail: Letters to the Editor may be sent to: Editor, Circuit Cellar INK, 4 Park St., Vernon, CT 06066.

Phone: Direct all subscription inquiries to (800) 269-6301. Contact our editorial offices at (860) 875-2199.

Fax: All faxes may be sent to (860) 871-0411.

BBS: All of our editors and regular authors frequent the Circuit Cellar BBS and are available to answer questions. Call (860) 871-1988 with your modem (300–14.4k bps, 8N1).

Internet: Letters to the editor may be sent to **editor@circellar.corn.** Send new subscription orders, renewals, and address changes to **subscribe@circellar.com.** Be sure to include your complete mailing address and return E-mail address in all correspondence. Author E-mail addresses (when available) may be found at the end of each article. For more information, send E-mail to **info@circellar.com.**

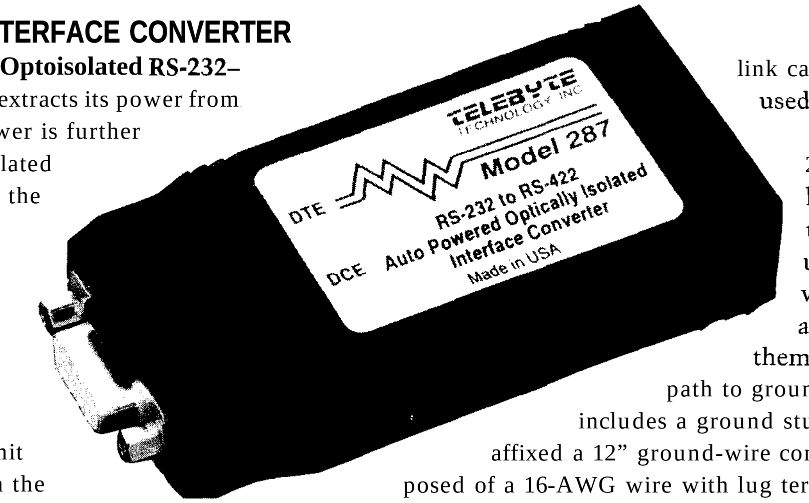
WWW: Point your browser to **http://www.circellar.com/.**

FTP: Files are available at ftp://ftp.circellar.com/pub/circellar/.

NEW PRODUCT NEWS

Edited by Harv Weiner

OPTICALLY ISOLATED INTERFACE CONVERTER


TELEBYTE TECHNOLOGY INC
Model 287
DTE
DCE
RS-232 to RS-422 Auto Powered Optically Isolated Interface Converter
Made in USA

The Model 287 miniature Optoisolated RS-232–RS-422 Interface Converter extracts its power from the **RS-232** interface. The power is further conditioned to provide an isolated power supply which operates the link side of the interface converter. Its isolation capability withstands 4-kV surges.

The Model 287 includes a DE-9 female connector, so it can attach directly to any PC COM port. A DTE/DCE switch can reverse the transmit and receive data signals when the Model 287 interfaces to a different type of port. The control signals are also used as a power source. Data is accommodated at rates up to 19.2 kbps.

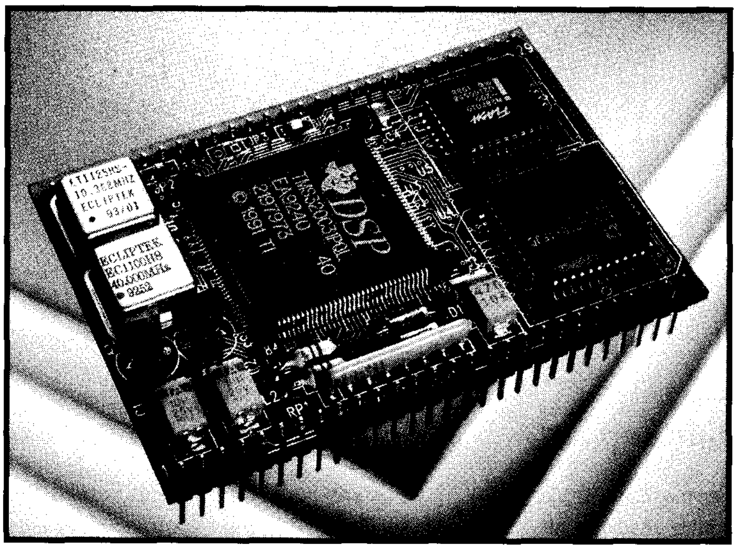
The link interface is complemented in an RJ-12 modular connector. The transmit and receive data signals use the four inner pins of the RJ-12 connector, while the two outside pins are reserved for grounding a shield on the link cable if one is used.

The Model 287 withstands high-voltage transients using circuitry which detects and directs them through a safe path to ground. The path includes a ground stud to which is affixed a 12” ground-wire connection composed of a 16-AWG wire with lug terminals.

The Model 287, which is housed in a small plastic container measuring 3.3” x 1.7” x 0.95″, sells for $155.

Telebyte Technology, Inc.
270 Pulaski Rd. • Greenlawn, NY 11740
(516) 423-3232 • Fax: (516) 385-8184
http://telebyteusa.com/

#500



SPEECH-CODING SUBSYSTEM

The Piranha from DSP Research is a complete speech-coding subsystem on a single plug-in module. The P3112 is driven at 60 MHz by a TMS-320C31 processor. It has a full-duplex analog interface, on-module A/D-D/A conversion with filters, an externally clocked serial interface, and 512 KB of zero-wait-state SRAM. The unit is ideal for DSP applications like speech compression and fax/modems.

The P3112 stores user programs in a single 128- or 256-KB EPROM. For voice-processing applications, DSP Research supplies a large library of ready-to-run algorithms, including ITU 6.728 Low Delay CELP, G.726 ADPCM, G.722 Subband ADPCM, 6.711 p-/A-law companding, USFS CELP 1016, and USFS LPC10e 1015. Algorithms can be selected by changing the serial clock rate or by setting mode bits. The speed and SRAM size enable the P3112 to support MIPS- and RAM-intensive applications such as modems, line- and acoustic-echo cancellation, and very low bit-rate voice compression.

You can evaluate the P3112 via the P31EVM evaluation platform using speech input from a taped source or telephone handset. Individual P3 112 modules cost $1250. The P31EVM, which comes with a single P3 112 module installed, is priced at $1995.

DSP Research, Inc.
1095 E. Duane Ave.,
Ste. 203
Sunnyvale, CA 94086
(408) 773-l 042
Fax: (408) 736-3451
http://www.dspr.com/

#501

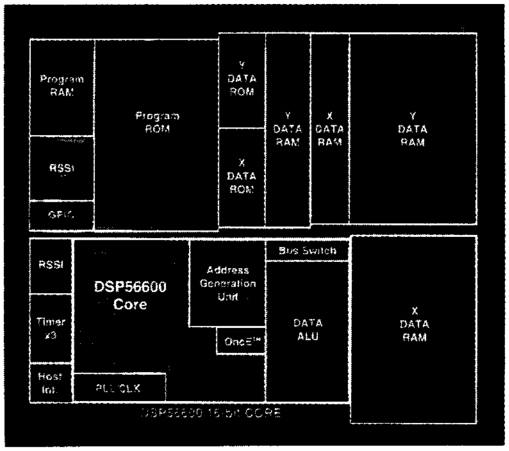
NEW PRODUCT NEWS

DSP FEATURES LOW POWER AND HIGH SPEED

The Motorola DSP56602 delivers 60 MIPS at 60 MHz and 2.7 V. It features a 24-KB program ROM along with 12 KB of data ROM divided between x and y memories. For RAM, the DSP56602 includes 4 KB each of x and y RAM and 5 12 words of program RAM.

It uses two Reduced Synchronous Serial Interfaces (RSSIs), which offer high-speed serial links to analog converters with different data rates. A triple timer module composed of a common 14-bit prescaler and three identical 16-bit timers is also provided. In addition, there are 3-3 1 general-purpose I/O (GPIO) lines available (depending on the peripherals enabled) and 3 external dedicated interrupt lines. An HI08 8-bit host interface provides glueless connection to other controllers, processors, and DSPs.

The DSP56602 offers on-chip program ROM patching. Service providers can correct or enhance software without returning the product to the manufacturer. The patch logic consists of four patch address registers. Writing an address to one of these registers starts a compare each time a new instruction is fetched from internal program ROM. A jump to a predefined location in on-chip RAM containing the correct or enhanced code replaces the compared instruction.



The DSP56602 offers a highly parallel instruction set which controls the Arithmetic Logic Unit (ALU), the Address Generation Unit (AGU), and the Program Control Unit (PCU). The data ALU features a fully pipelined 16-x16-bit parallel multiply accumulator with a 40-bit parallel barrel shifter for single clock-cycle throughput.

It also includes a phase-locked loop that enables the core to run at full speed from low-frequency, off-chip clock sources, even though they are at a much lower frequency. The enhanced OnCE module lets developers examine and modify all internal resources in real time, reducing software-development efforts.

The DSP56602 costs less than $20 in quantity.

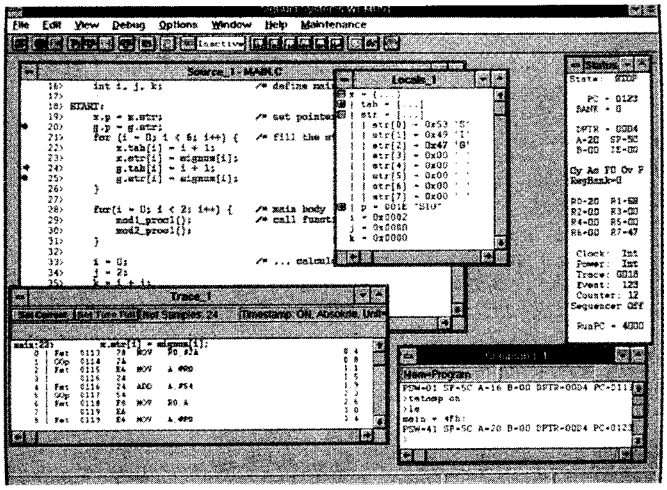
**Motorola DSP Division
MS 0E314
6501 William Cannon Dr. W
Austin, TX 78735-8598
(512) 891-2000
Fax: (512) 891-3877
http://www.motorola-dsp.com/**

#502

8051 DEVELOPMENT ENVIRONMENT

Signum Systems has released a Windows-based version of its development environment. The Embedded Systems Development Studio ESDS-51 operates with the USP-5 1 emulator, a state-of-the-art in-circuit emulator that supports most popular 805 1 -based microcontrollers from Intel, Philips, Dallas Semiconductor, and others. ESDS-51 is available bundled with the USP-5 1 emulator and as an upgrade for previously purchased systems.

ESDS-5 1 runs under Windows 3.1, Windows 95, and Windows NT. It supports all major C- and PL/M-5 1 compilers at the source level with the latest advanced software debugging and verification tools (e.g., variable quick watch, pass-points, performance analysis, coverage analysis, and register breakpoints).



```
    int i, j, k;                    /* define mai
START:
    x.p = x.str;                    /* set pointe
    g.p = g.str;
    for (i = 0; i < 6; i++) {       /* fill the st
        x.tab[i] = i + 1;
        x.str[i] = signum[i];
        g.tab[i] = i + 1;
        g.str[i] = signum[i];
    }

    for(i = 0; i < 2; i++) {        /* main body
        mod1_proc1();               /* call functi
        mod2_proc1();
    }

    i = 0;                          /* ... calcul
    j = 2;
    k = i + i;
```

```
x = {...}
| tab = [...]
| str = [...]
| | str[0] = 0x53 'S'
| | str[1] = 0x49 'I'
| | str[2] = 0x47 'G'
| | str[3] = 0x00 ' '
| | str[4] = 0x00 ' '
| | str[5] = 0x00 ' '
| | str[6] = 0x00 ' '
| | str[7] = 0x00 ' '
| p = 001E "SIG"
i = 0x0002
j = 0x0000
k = 0x0000
```

```
State:  STOP

  PC = 0123
BANK = 0

DPTR = 0004
A=20   SP=5C
B=00   IE=00

Cy Ac F0 Ov P
RegBank=0

R0=20  R1=6B
R2=00  R3=00
R4=00  R5=00
R6=00  R7=47

 Clock:  Int
 Power:  Int
 Trace:  0018
 Event:   123
 Counter: 12
Sequencer Off

 RunPC = 4000
```

```
main:23>        x.str[i] = signum[i];
  0 | Fet  0113  78  MOV  R0,#2A
  1 | OOp  0114  2A
  2 | Fet  0115  E4  MOV  A,#R0
  3 |      0116  24
  4 | Fet  0116  24  ADD  A,#54
  5 | OOp  0117  54
  6 | Fet  0118  F8  MOV  R0,A
  7 |      0119  E4
  8 | Fet  0119  E6  MOV  A,@R0
```

```
PSW=01 SP=5C A=16 B=00 DPTR=0004 PC=0111
>tstamp on
>ls
main + 4Fh:
PSW=41 SP=5C A=20 B=00 DPTR=0004 PC=0123
>
```

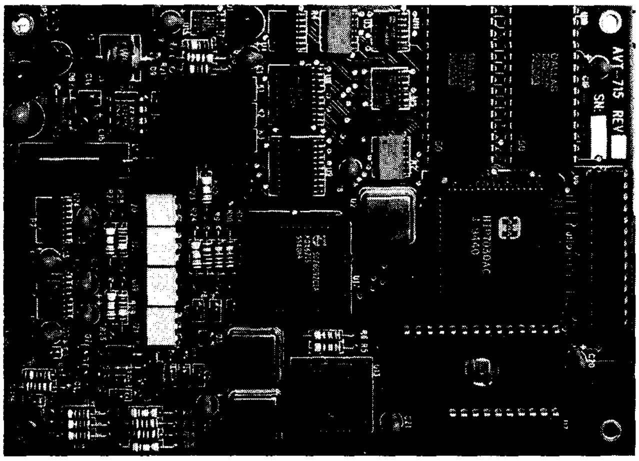
ESDS-5 1 features a cycle-level simulator for debugging without an emulator. It also supports remote debugging via IP networks. Remote debugging enables an ESDS-5 1 session to be accessed over the network using a standard telnet client for co-debugging or customer support.

**Signum Systems
11992 Challenger Ct.
Moorpark, CA 93021
(805) 523-9774
Fax: (805) 523-9776
http://www.signum.com/**

#503

NEW PRODUCT NEWS

DUAL J1850 INTERFACE



The AVT-715 is a dual J1850 interface that enables a PC to connect to and communicate with a J1850 bus-equipped automobile. The SAE J1850 is a two-part standard for an in-vehicle multiplex bus that enables various sensors, actuators, and controllers to communicate. Additionally, external test equipment can access the bus for diagnostics and other maintenance purposes.

The AVT-715 provides both a Variable Pulse Width (average bit rate = 10.4 kbps) and Pulse Width Modulation (average bit rate = 41.6 kbps) interface on one board. Functionally, the AVT-715 acts as a protocol translator and message buffer between a controlling PC and the J1850 bus. Communications between the AVT-715 and the PC is via an RS-232 or RS-422 serial link (jumper selected). The serial data rate is jumper selected among four baud rates (9.6, 19.2, 38.4, and 57.6 kbps).

The AVT-715 is offered as a ready-to-use package for $1200 or as an OEM module for $800. The ready-to-use package includes the AVT-715 dual J1850 interface board housed in a NEMA 4X enclosure, a serial interface cable, an OBD-II cable assembly, Windows 3.1x controller software, and a user manual. The OEM module consists of the board and user manual with programmer's information.

Advanced Vehicle Technologies, Inc.
1509 Manor View Rd.. Davidsonville, MD 21035
(410) 798-4038 . Fax: (410) 798-4308

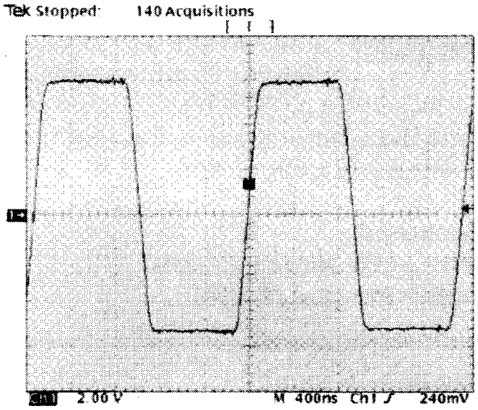
#504

NEW PRODUCT NEWS

RS-232 TRANSCEIVER

The MAX3237 high-speed data transceiver is ideal for fast modem applications. Dual internal charge pumps and a proprietary, low-dropout output stage ensure true RS-232 output levels for data rates to 1 Mbps and above. The charge pumps require only four small external capacitors.

In normal operating mode with a worst-case load of 3 kΩ and 1000 pF, the MAX-3237's guaranteed data rate (250 kbps) makes it compatible with PC-to-PC communication software such as Laplink. In its megabaud mode (MBAUD = v,.) with a maximum load of 3 kΩ in parallel with 250 pF, the guaranteed data rate is 1000 Mbps with guaranteed slew rates of 24 V/µs.



The MAX3237 contains five drivers and three receivers, providing a complete serial port for notebook, subnotebook, and palmtop computers. For modem support, it offers a 1-µA shutdown mode in which all three receivers remain active. This capability enables the MAX3237 to monitor modems and other external devices without the danger of heavy current (in the event that V_{CC} to an external device is turned off) resulting from forward bias on a protection diode in that device.

The MAX3237 comes in a 28-pin SSOP and costs $3.29 in quantity.

Maxim Integrated Products
120 San Gabriel Dr.
Sunnyvale, CA 94086
(408) 737-7600
Fax: (408) 737-7194
http://www.maxim-ic.com/

#505

NEW PRODUCT NEWS

A/D ADAPTER

ADNAV Electronics has announced the **LPT:*Analog! V.2.0*, a** low-cost 12-bit A/D adapter for the PC parallel printer port. The adapter is a simple and convenient interface for acquiring analog signals directly on a PC at rates of up to 7500 samples per second. Signal-conditioning information is supplied, making it easy to collect data from analog-output instruments, sensors, 4–20-mA loops, and many other analog sources.

Power for the unit is derived directly from the PC's printer port. The micropower design of the adapter's internal circuitry ensures that it can be used with virtually any desktop, laptop, or notebook PC. The adapter's design, in conjunction with its small size-a shielded case measuring 2" x 2.4" x 0.7"—makes it ideal for PC-based data-acquisition applications that require ultimate portability and flexibility.

LPT:*Analog! V.2.0* comes with software utilities that enable real-time display of acquired waveforms and ASCII-format data logging to disk, as well as high-resolution spectral analysis of acquired data. The source code (QBasic, C, and assembly) for these utilities is included at no extra cost. Software drivers, Visual Basic DLLs, and programming examples are also provided. The adapter sells for $59.

ADNAV Electronics
58 Chicory Ct. • Lake Jackson, TX 77566
(409) 292-0988 • Fax: (409) 292-0988

#506

PC BUS MOTION CONTROLLER

The **Model 5643 Intelligent IndustryPack (IP) Carrier Board,** featuring a TI '486SLC CPU running ROM-DOS, lets you develop motion-control and other applications using standard PC-based development tools. DSP-based motion processors can be added in a modular fashion to the board.

Acting as a coprocessor on the EISA bus, the 5643 ensures that real-time control functions are not degraded by the host CPU overhead or operating-system limitations. Hence, systems using a Windows-based GUI can reliably offload machine-control functions to the 5643, which operates independently from host-PC activities.

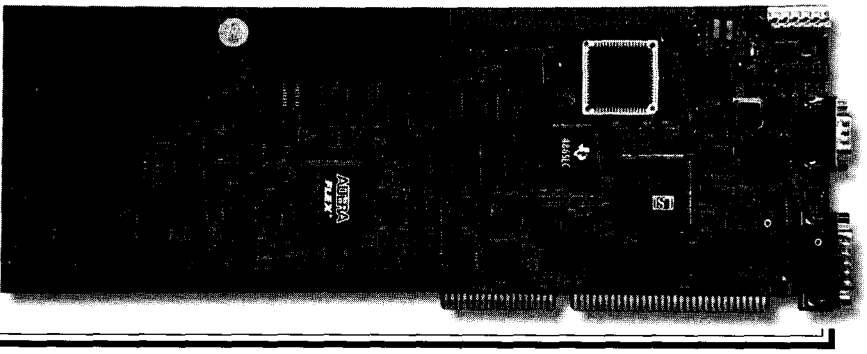
The 5643 features 32-KB battery-backed static RAM for user-defined parameter storage and 16-KB dual-port RAM for fast communication with the host PC. It has a 2-, 4-, or 6-MB DRAM configuration for running large programs and an optional 2 MB of flash memory for additional program storage.

Also available are 48 programmable DIO and an optional math coprocessor for applications that demand floating-point calculations. Dual serial ports (RS-232 and RS-232/-485) provide fast communications with external systems.

The board has sockets for four single- or two double-wide IP modules which can be I/O or memory mapped. The 5643 was created for up to 8 axes of servo and/or stepper motion control. The board's modular design allows ADC and DAC control, encoder interfacing, and other functions to be incorporated with motion control. A PC/104 interface offers additional expansion capabilities.

Standard power and utility connectors give the 5643 stand-alone capabilities. A keyboard is easily implemented in the 10-pin utility interface.

The board's software library is compatible with most C, C++, Visual Basic, Visual C++, and Turbo Pascal compilers, and it includes Windows DLLs. Tech 80's Modular Motion toolkit lets designers easily incorporate IP modules onto the 5643 using a Windows-based interface. Pricing for the Model 5643 begins at $1450. Also available is the 5643 Development Kit for easy implementation in development and prototype stages.



Technology 80, Inc.
685 Mendelssohn Ave. N
Minneapolis, MN 55427
(612) 542-9545
Fax: (612) 542-9785

#507

FEATURES

FEATURE ARTICLE

Bob Fine

Building a High-Performance DSP System

Bob introduces Analog Devices' new ADSP-2100 family. After reviewing architectural characteristics, he uses a FIR-filter system design to demonstrate how easy it is to get a DSP up and running.

DSP technology is becoming more pervasive as access to DSPs and development tools increases and costs come down. As designers become more aware of DSP performance benefits, they're creating new ways of implementing DSP in their systems.

For those not familiar with some of the DSP devices on the market, I'd like to introduce you to Analog Device's ADSP-2100 family of DSPs. 1'11 look at the ADSP-21*xx* architecture, walk through a system-design example, and discuss issues relating to both hardware and software development.

You should get enough information to get started on a DSP system design.

FAMILY ARCHITECTURE

All ADSP-21*xx* DSP chips share a common base architecture. As shown in Figure 1, this architecture consists of arithmetic, address-generation, and program-sequencer sections.

The architecture is rich with registers and flexible in performing calculations and moving data. The internal bus structure also provides flexibility in data and instruction handling.

Several features of DSP architecture set it apart from conventional microprocessor or RISC architecture. Most notably, DSP performs fast, single-

cycle arithmetic operations. To ensure that the processor sustains these calculations every cycle, the required data operands are fetched every cycle.

The arithmetic, data-addressing, and program-sequencing sections of the ADSP-2100 base architecture are interconnected via a modified Harvard architecture bus structure.

A modified Harvard architecture supports an instruction fetch from program memory in the same cycle as two data fetches take place-one from data memory and the other from program memory. The ability to fetch two data words and one instruction in a single cycle is critical for efficient implementation of DSP algorithms.

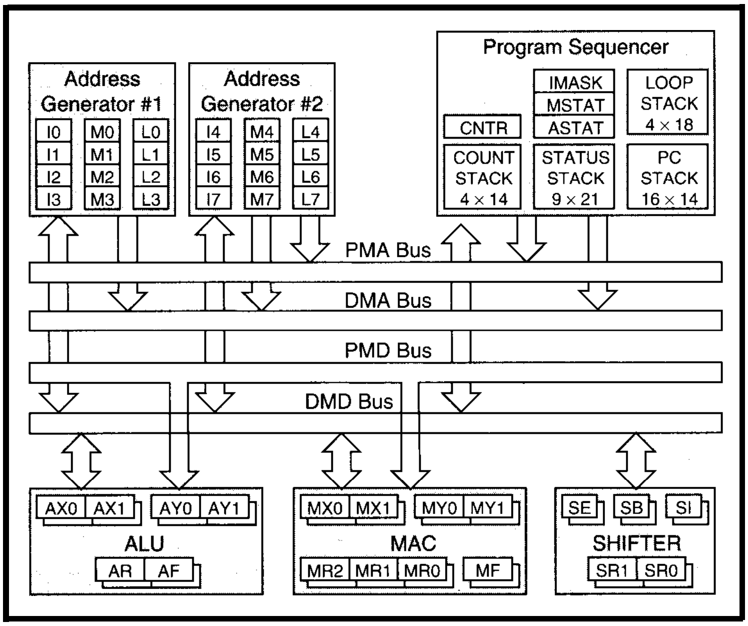


Figure 1—*The* **base architecture for** *the ADSP-2100* **family consists of** an **arithmetic section containing an ALU, MAC, and** *shifter.The16-bit arithmetic* **units** *a/so perform* **multiprecision operafions** *for larger* ***data*** *words. Two independent data-address generafors* ***support*** *zero-overhead circular* ***modulo*** *addressing, and a program sequencer supports zero-overhead nested looping.*

ARITHMETIC SECTION

Like a digital filter, many DSP algorithms have arithmetic operations performed with data and coefficient values. The key advantage of DSPs over other micros is the ability to perform arithmetic operations in a single cycle.

Responding to interrupts in real time also involves the arithmetic-section architecture. If the arithmetic unit is used in the interrupt service routine, the data used in the main routine needs to be saved so it can be restored after the interrupt is serviced.

This technique is called a context save and restore. A DSP with an automatic context-switching mechanism is the most efficient for real-time signal-processing applications.

The arithmetic section consists of a 16-bit ALU, a 16-bit MAC with a 40-bit accumulator, and a general-purpose shifter. Each arithmetic unit has a number of input and output registers, interconnected by the bus structure, which consists of data-memory, program-memory, and result buses. Through this bus structure, two operands can be fed to the arithmetic units in a single cycle.

The 24-bit-wide instruction and the instruction set of the ADSP-21*xx* processors allow a high level of parallelism. Many operations occur in one cycle, yielding a high MOPS rating.

DATA ADDRESSING

DSP algorithms require two operands fetched from memory in a single cycle to be subsequently fed to the arithmetic units. To supply the addresses of these two operands in a flexible manner, two address generators are required.

Given the modified Harvard architecture, one address generator must supply an address over the data-memory address bus while the other supplies an address over the program-memory address bus.

DSP algorithms also require circular buffering (i.e., data in a buffer is addressed so the address pointer wraps from the end back to the start of the buffer). Some applications need to support circular buffering even when the pointer advances by values greater than one. Figure 2 shows how this modulo circular buffering works.

The address-generation circuitry performs address modify-and-compare operations in hardware for optimum efficiency. Performing these functions in software limits the processor's ability to handle real-time signals.

In Figure 2, a buffer of eight locations resides in memory starting at address 30. The address generator calculates the next addresses within the buffer while keeping the proper data spacing so two locations are skipped.

The address generator outputs address 30 onto the address bus while it modifies the address to 33 for the next cycle's memory access. When address 36 is modified to address 39 (outside the buffer), the address generator detects that the address is outside the buffer boundary and modifies the address to 3 1. It is therefore connected back into the start of the buffer.

The update, compare, and modify occur with no overhead. In one cycle, address 36 is output onto the address bus. In the next cycle, address 31 is output onto the address bus.

The data-addressing section of the ADSP-21*xx* architecture has two address generators, each containing four address (I), four modify (M), and four length (L) registers.

An address is output to the address bus by specifying an I register. The

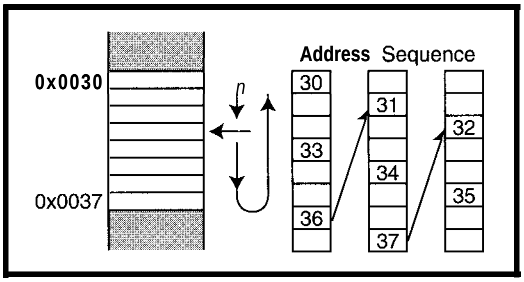


Figure *2-Jhe data-address generators within the DSP must be able to support* ***modulo*** *circular buffering. In* ***this*** *example,* ***the*** *address is modified by* ***the*** *value 3 in each instruction* ***cycle.This*** **modification is performed** ***with*** *zero* **overhead so a** *new* **address can** *be* **output every cycle.**

address is modified in preparation for the next operation by the specified M register. The L register specifies the length of a circular buffer.

PROGRAM SEQUENCING

The program sequencer must loop through algorithm code efficiently without incurring any extra time or overhead to get from the end to the start of the loop. This so-called zero-overhead looping is quite different from conventional microprocessors.

A typical microprocessor requires you to maintain program loops in software. A conditional instruction placed at the end of the loop determines whether a j ump directs program flow back to the top.

With a DSP, the loop's last instruction executes in one cycle. In the next cycle, either the first instruction at the top or the first one outside the loop is executed. Of course, hardware support for nested loops is required, especially for multidimensional algorithms.

DSP algorithms also involve real-world signals converted into digital representations by the ADC. The ADC usually lets the DSP know that a data sample is ready via a hardware interrupt. Since interrupts need to be handled in real time in a DSP application, the DSP must respond to interrupts quickly and efficiently.

Device	Voltage	Instruction Rate (MIPS)	Program Memory RAM (words)	Data Memory RAM (words)	Comments
	5				
ADSP-2101	3.3	25	2 K	1 K	2 serial ports
ADSP-2103		10	2 K	1 K	2 serial ports
ADSP-2104	5	20	512	256	2 serial ports
ADSP-2104L	3.3	13.8	512	256	2 serial ports
ADSP-2105	5	20	1 K	512	1 serial port
ADSP-2109	5	13.8		256	4 K word ROM, 2 serial ports
ADSP-2109L	3.3			256	4 K word ROM, 2 serial ports
ADSP-2115	5	25	1-K		2 serial ports
ADSP-2161	5	16.6		512 512	8 K word ROM, 2 serial ports
ADSP-2162	5	10		512	8 K word ROM, 2 serial ports
ADSP-2163	3.3	.10		512	4 K word ROM, 2 serial ports
ADSP-2164		33		512	4 K word ROM, 2 serial ports
ADSP-2171	5		2 K	2 K	2 serial ports, host port
ADSP-2172	5	33	2 K	2 K	8 K word ROM, 2 serial ports, host port
ADSP-2173	3.3	20	2 K	2 K	2 serial ports, host port
ADSP-2181	5	33	16 K	16 K	2 serial ports, 2 DMA ports
ADSP-2183	3.3	28.8	16 K	16 K	2 serial ports, 2 DMA ports
ADSP-2185	5	33	16 K	16 K	2 serial ports, DMA porta
ADSP-2185b	3.3	25	16 K	16 K	2 serial ports, DMA ports
ADSP-2186	5	33	8 K	8 K	2 serial ports, DMA ports
ADSP-2186L	3.3	25	8 K	8 K	2 serial ports, DMA ports
ADSP-21 msp58	5	26	2 K	2 K	host port, analog I/O
ADSP-21msp59	5	26	2 K	2 K	4 K word ROM, host port, analog I/O

Table l-Analog Devices *has one of the largest code-compatible families of DSPs. A single development system lets you develop code for* ***all the*** *processors listed, so you can migrate from one* device to *another while preserving code.*

The ADSP-21*xx* program sequencer determines the next instruction to be fetched from program memory. It supports nested and zero-overhead loops, multiple stacks, and low-latency interrupt responses. Many sequencing functions are performed in the hardware rather than in the software.

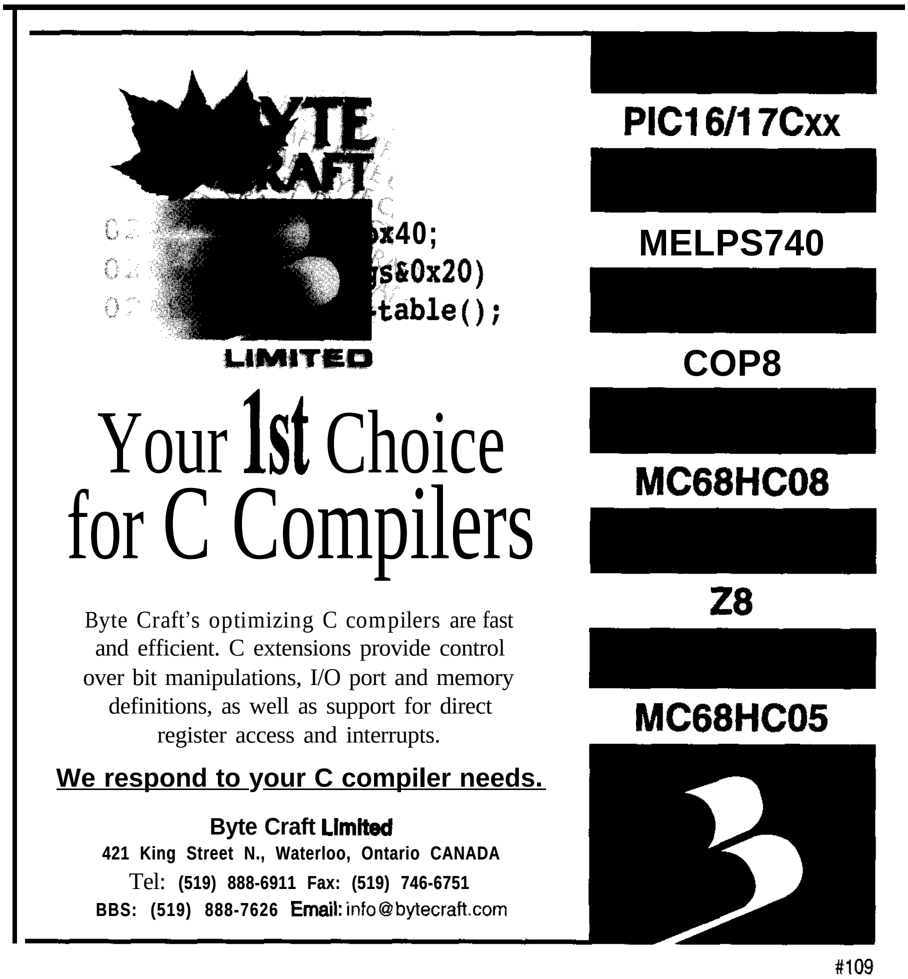
ASSEMBLY-LANGUAGE SYNTAX

The assembly-language syntax of the ADSP-2100 family is algebraic rather than mnemonic. As you can see, an arithmetic operation and two data fetches are specified in a single instruction for single-cycle execution:

```
MR = MR + MXO * MY1 (SS),
  MXO = DM(IO, M2),
  MY1 = PM(I5, M4);
```

The first clause instructs the processor to multiply the data in register MXO with the data in register MY 1. This product is added to the contents of the multiplier result register, MR, and stored as the new contents of MR.

The second clause instructs the processor to fetch data from data memory using the contents of the register IO as the address. While data is being fetched and stored in register MXO, the address is modified by the contents of register M2 and reloaded into IO. If length register LO was loaded with a nonzero value, the update is performed, keeping the address value within a circular buffer.



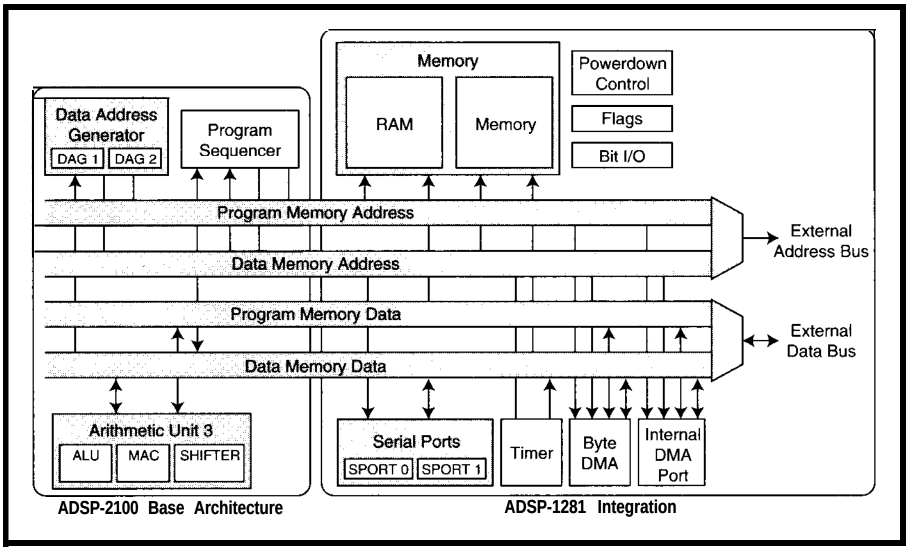


Figure 3-The *ADSP-2181* **adds** *16* **K** *words* **ofprogram-memory RAM, 16** *K words of* **data-memory RAM, two** *serial ports with DMA* **capability, a programmable timer,** *an 8- and 16-bit DMA pod, power-down control logic, and bit I/O. The large amount of on-chip RAM eliminates the need for* **external RAM,** *saving design time, cost, and board space.*

The third clause, like the second clause, loads the register MY1 with a value from program memory while updating the address pointer IS.

ADSP-2100 FAMILY

The ADSP-2100 family starts with low-cost devices under $5 and continues up to high-performance, highly integrated devices. The members differ in memory and peripheral configurations.

In some cases, different members have identical pinouts, so a hardware upgrade can be performed by plugging a new DSP into the existing socket. This feature provides a low-cost entry point with an upgrade path leading to higher performance, larger memory, and more peripherals without changes to the circuit board.

Table 1 summarizes the ADSP-2 lxx devices and their key features. All devices have a 16-bit timer with a prescaler, boot address generator for automatic booting from an external EPROM at powerup with no glue logic, and an external parallel port.

DEVELOPMENT TOOLS

Simulators let you profile the code's performance, plot digital signals on the screen, simulate I/O capabilities, and set complex breakpoints. A system builder creates a description of the hardware system (e.g., how much memory and what extra components are used) to understand what the system looks like and flag errors. After the code is debugged, the PROM splitter formats the DSP code for download to a PROM programmer.

Analog Devices' software development package supports all fixed-point processors. It contains the assembler, system builder, linker, simulator, PROM splitter, a C compiler and source-level debugger, C runtime library, and librarian.

Once the DSP program is loaded onto the processor and the hardware is running, EZ-ICE can provide nonintrusive target-based debugging. It uses a wide range of functions, including single-step and full-speed execution with predefined breakpoints, and it can view and change contents of the processor's registers and memory.

EZ-Lab evaluation boards have small, low-cost controls (e.g., push buttons and indicator lights) and an EPROM socket for stand-alone operation.

ADSP-2181

The ADSP-218 1 offers the highest level of integration available in a 16-bit, fixed-point DSP. It contains 16 K x 16 words of data memory and another 16 K x 24 words of program memory for instruction and data storage.

Data-memory addressing is controlled by two programmable address generators. The address generators can

point to data words in data memory as well as program memory.

A program sequencer controls instruction execution and maintains the address for instructions in the program memory. The ADSP-2181's arithmetic section consists of a MAC, an ALU, and a shifter. Two serial ports, an interval timer, and interface circuitry complete the ADSP-2181's architecture shown in Figure 3.

With a highly integrated DSP, the memory subsystem is built in, resulting in improved data flow. DSPs can only access external memory once per cycle because they typically have a single external bus.

The ADSP-2 18 1 performs three internal memory accesses each cycle. In 30 ns, the next instruction can be fetched while one data word is fetched from data memory and another from program memory.

G.721 Wide-Band ADPCM	26.8 &sample
G.728 LD-CELP	113.6 µs
GSM Speech Coding	1.975 µs
Half-Rate GSM Speech Coding	13 ms
IS-54 VSELP Speech Coding	9.78 ms
1024-Point Complex FFT	1.07 ms
4096-Point Complex FFT	5.5 ms
Full-Duplex Speaker Phone	9.5 MIPS
V.32bis modem	15 MIPS
V.34bis modem	28 MIPS

Table 2—*The ADSP-2181, with* its *multi-function* instructions, *performs many operations in a singfe instruction cycle. This feature allows the ADSP-2181 to execute* sophisticated *DSP* algorithms in fewer *MIPS fhan other DSPs.*

The ADSP-2 18 1 is a register-based processor with registers surrounding the inputs and outputs of all the arithmetic units. These registers further enhance the processor's ability to efficiently move and store data.

The large internal memory and efficient architecture of the ADSP-2181 translate into performance not possible on other fixed-point processors. The best example is the 4000-point FFT which executes on the ADSP-2181 in 5.5 ms with no external memory.

ACCESSIBLE MEMORY?

But, on-chip memory doesn't solve anything unless it's accessible. The ADSP-2181 has several ways to directly access the memory-serial port DMA, byte memory DMA (BDMA), and internal memory DMA (IDMA).

These DMA channels allow data to be transferred in and out of the ADSP-218 l's internal memory while your program is running. When the transfer is complete, the processor's internal circuitry generates an interrupt to let the program know that the memory transfer has taken place.

OTHER SYSTEM CONSIDERATIONS

Do you need complex algorithms requiring high computational throughput from the DSP! Are you designing battery-powered equipment or a system that has low power consumption?

The benefits of large memory and the variety of DMA features are further enhanced by a 33-MHz instruction rate, an enhanced instruction set, and low-power modes.

The instruction rate deserves a closer look because rates alone do not indicate a processor's performance. It's better to look at how many operations can be performed in a single cycle.

In a single cycle, the ADSP-218 1 can:

- fetch a data word from data memory
- calculate the next address for data memory with a user-specified value
- fetch a second data word from program memory
- calculate the next address for program memory with a user-specified value
- detect the passing of a circular buffer boundary and automatically correct the memory pointer for both data and program memories
- fetch the next instruction from program memory
- detect the termination of an instruction loop and automatically adjust the program counter to either the start of the loop or outside the loop
- perform an arithmetic calculation with user-specified source registers
- store the result of the arithmetic operation into a specified destination register
- update the timer
- receive and transmit data through the two serial ports

This cycle runs every 30 ns at the 33-MHz rate.

This code for the core loop of an FIR filter illustrates the high degree of parallelism in the instruction set:

```
DO TAP-LOOP UNTIL CE
TAP-LOOP: MR=MR+ MX0*MY1(SS),
   MX0=DM(I0,M2),
   MY1=PM(I4,M5);
```

The value in register MXO is multiplied by the value in register MYl. This product is added to the result-register contents and stored as the new contents of the result register.

The registers MXO and MY l are also loaded from the data memory and program memory, respectively. The address pointers IO and 14 are modified by the values in registers M2 and M5, respectively. At the same time, the next instruction in program memory is fetched and decoded.

The external clock only needs to be half the instruction rate. To run the processor at 33 MHz, connect it to a 16.5-MHz crystal or oscillator.

Because of the low speed, the crystal oscillator or clock is easier to find and less expensive to buy. And, you won't have to worry as much about the effects of radiated high-frequency noise.

Many applications require a DSP with low power consumption. The ADSP-218 l's low-power modes extend battery life, decrease heat caused by power-supply-regulator power dissipation, and decrease radiated noise levels from high-frequency current surges.

SYSTEM COMPARISON

Using an audio signal-processing system design, I implemented conventional and highly integrated DSPs. I'll compare the cost, design time, and performance of the systems.

The following DSP system contains 32 K words of static RAM, a host interface, and a boot ROM. Since the DSP doesn't have enough internal RAM, external RAM is added. The access time required for these RAMs is 12 ns.

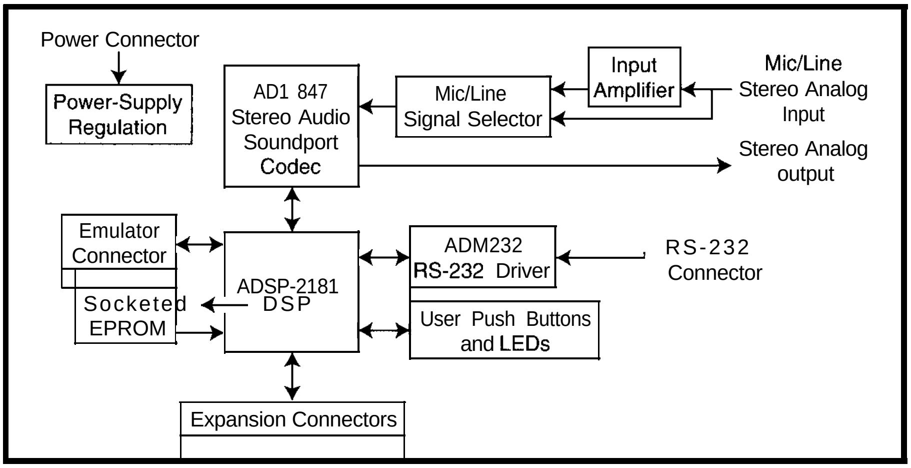


Figure 4—*EZ-Kit Life is designed around the ADSP-2181 DSP processor. An AD1847 stereo audio codec provides the A/D and D/A conversion and analog-signal interface. An RS-232 interface connects the kit board to a computer or any system supporting RS-232. User push buttons, LEDs, a socketed EPROM, and expansion connectors simplify prototyping and experimentation.*

An address-decoder circuit creates the various chip-select signals for the RAM components. The address decoder requires -3 ns of propagation delay from the time the address on the bus settles to the time the outputs are valid.

The 12-ns access time of the RAM plus the 3-ns address decode time yields a total access requirement of 15 ns. The host processor connects to the DSP system bus by an interface circuit. It can also access the DSP RAM via the DMA controller circuit.

The DSP system based on the ADSP-218 1 contains 32 K words of on-chip RAM (external memory is unnecessary). Since the complete memory system is on-chip, an address decode circuit or timing analysis isn't required.

The host connects to the DMA port of the ADSP-2181 with an interface circuit.

Since the DMA capabilities of the ADSP-2181 are used, an external DMA controller isn't required. The boot EPROM connects to the byte DMA port and combines with byte RAM, yielding 4 MB of additional memory space-without extra interface circuitry.

Depending on the cost of the memory components, a system using a highly integrated DSP can cost $20 less. Eliminating the memory subsystem design and related test and debug can shave weeks off the development schedule.

PERFORMANCE BENCHMARKS

Since processors' performances vary drastically-even with the same clock rate-it's best to look at benchmarks of algorithms you're going to run in your system and operations per cycle.

Judging a DSP by its clock speed or instruction rate alone can be misleading. For example, a processor with a 40-MIPS rating that performs only one operation per instruction is less powerful than a 20-MIPS processor performing three operations per instruction.

Processor bandwidth benchmarks are usually specified as a percent loading or a MIPS requirement. So, if a 20-MIPS processor implements a particular benchmark using 10 MIPS, the processor is 50% loaded.

It may be possible to run two channels of an algorithm or implement two algorithms simultaneously. Table 2 lists the benchmarks for the ADSP-2181-based systems.

EVALUATION AND DEVELOPMENT SYSTEM

The ADSP-218 1 is supported by the low-cost EZ-Kit Lite development system. It includes the EZ-Lab evaluation and development board and the software development package (see Figure 4).

The ADSP-218 1 EZ-Lab provides a good example of an audio-processing system. The board features the 33-MIPS ADSP-2181, the AD1847 stereo audio SoundPort codec, an RS-232 interface based on Analog Devices' ADM 232 chip, and an analog input-signal-conditioning circuit based on the Analog Devices' SSM2135 dual op-amp. This onboard codec provides CD-quality stereo with a programmable sampling rate from 5.5 to 48 kHz.

The board connects to the RS-232 or COM port of a PC. The board can also be configured to run in standalone mode with programs you've developed.

Once the board is connected to a power supply, a PC's COM port, an analog input source (e.g., microphone, CD player, tape player, etc.], and a set of powered speakers, you're ready to install the software.

STARTING OUT WITH DSP

Building a high-performance DSP system with a member of the ADSP-2100 family gives you a low-cost entry point to DSP with an upgrade path to high-performance, highly integrated processors.

Since all ADSP-2100 family members are based on the same core architecture, they are code compatible. This means you don't have to recode your algorithm when moving from one processor to another. And, with low-cost development tools, it's easy to get up and running quickly. ▣

Bob Fine is the product line manager for Analog Devices' fixed-point DSP products. He has over 10 years of DSP-system-design experience and has published a number of articles on DSP-design issues. You may reach Bob at bob.fine@analog.com.

SOURCES

ADSP-21xx, EZ-ICE, EZ-Kit Lite, EZ-Lab, datasheets
Analog Devices
P.O. Box 9106
Norwood, MA 02062-9106
(617) 329-4700
Fax: (617) 329-1241
http://www.analog.com/

I R S

401 Very Useful
402 Moderately Useful
403 Not Useful

FEATURE ARTICLE

Matt Park &
Brian McLeod

Real-Time DSP Modems with a PC and Sound Card

We've found yet another use for that sound card hidden deep inside your PC. This time, Brian and Matt explore real-time DSP-based V.22 QPSK and V.22bis modem demodulators using C++.

In "Real-Time DSP with a PC and Sound Card" (*INK* 61), a Windows-compatible sound card and a '486 PC were all that was needed for real-time DSP.

Building on that article, we offer examples of real-time DSP-based V.22 QPSK and V.22bis QAM modem demodulators. We also present some DSP building blocks you can use.

You might ask-why design something when you can buy a chip set with 20 times the performance? Because:

- demodulators are great DSP learning projects. They incorporate fundamental DSP building blocks such as IIR and FIR filters, down sampling, frequency mixing and shifting, adaptive equalizers, feedback loops, and real-time processing constraints.
- many DSP building blocks are useful for embedded systems control
- as embedded controllers and DSPs merge, you'll implement data communications directly on the controller, reducing chip count and cost

DIGITIZE32

V22Modem-Digitize32 demodulates V.22 1200- and 2400-bps modems in real time on a '486/33-MHz or faster PC. We wrote Digitize32 using VC++4.0 for Windows 95 and Windows NT.

Digitize32 is an upgrade of the 16-bit program in real-time spectral displays using Windows' low-level audio functions in INK 61. Most of its Windows code can be directly compiled with VC++1.5 (16 bit) or VC++4.0 (32 bit).

Most C++ compilers should compile the DSP functions and C++ classes.

Ctt OBJECTS

C++ is good for designing reusable DSP code. We used it over standard C because many of the functions (e.g., filters, time averagers, and storage arrays) were used more than once in the demodulator, and each instance required its own internal storage.

The C++ class and object concept enabled us to almost "wire" the objects together to build a very modular DSP application.

For example, with the filters in the demodulator, C++ let us write one filter class and, in the program, create two unique instances of that filter class.

But, this isn't a tutorial on C++. For that, see Mike Podanoffsky's "A C++ Programming Tutorial" (INK 57).

SIGNAL MODULATION

Quadrature Amplitude Modulation (QAM) uses a digital signal to modulate a carrier's amplitude and phase. QAM is well-suited for transmitting digital information over telephone lines and other high-quality, band-limited communications paths. QAM systems are coherent. They require the receiver to be phased locked with the transmitter's carrier and symbol clock.

QAM systems are called M-ary *QAM*, where *M* is the number of unique amplitude and phase combinations. Two values--l and Q-represent any M-ary QAM signal because any waveform can be expressed as a linear combination of orthogonal phasors.

The numbers of unique amplitude and phase combinations are called symbols. When plotted on an *IQ* or xy plot, they form constellations. Photo 1 shows the constellation for V.22bis.

Assigning multiple bits per symbol makes the symbol rate less than the data rate, reducing the required transmission bandwidth. This reduction is why we use the more complicated coherent QAM modulation.

The data rate measures how fast the computer sends bits out of its serial port (e.g., 1200 bps). The symbol (baud) rate is the rate that the modem changes the transmitted waveform (e.g., 600 symbols per second [sps]).

Phase-shift keying (PSK) is a subset of QAM where the digital signal modulates the carrier's phase. The CCITT standard specifies V.22 as a differential quadrature PSK (DQPSK) signal. For DQPSK, the phase difference between two symbols-not the phase value— encodes the symbol information.

V.22's600-sps symbol rate is one half the 1200-bps data rate. Each symbol is assigned two bits or a dibit. A 0 degree change is 01, 90 is 00, 180 is 10, and 270 equals 11.

Both V.22 and V.22bis use 1200 Hz for the low-channel carrier and 2400 Hz for the high channel. They have similar frequency spectra and channel bandwidths. Because they share so many characteristics, it was easy to reuse most of the V.22 DSP code for the V.22bis demodulator.

DEMODULATORS

V.22 modems-and most QAM systems-scramble transmitted data prior to the transmission modulation to get a pseudorandom datastream. Random data gives a constant frequency spectrum and power density and decreases the number of 1s or 0s in a row. It's thus easier to bit synchronize to the datastream.

Scrambling the data also makes each symbol equally likely and locates the optimum symbol-decision thresholds halfway between each symbol.

All demodulators must input an analog signal and output the best estimate of the received bits. Demodulators contain signal-to-levels, levels-to-bits, and bit-synchronizer (bit-sync) blocks as seen in Figure 1.

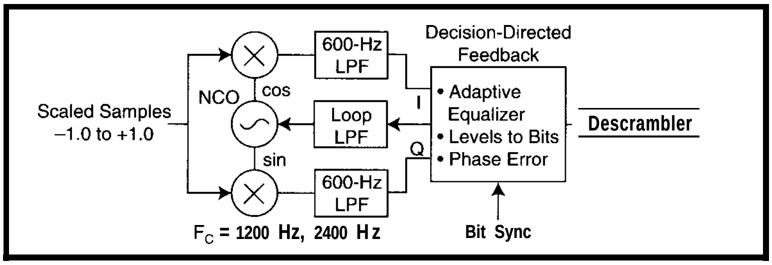


Figure I--These *are the basic building blocks of any QAM demod. The NCO, mixers, and LPF filters convert the signal to levels. The LPF filters can be implemented as either FIR or IIR filters. Adaptive equalizers are optional for some QAM demods. The external bit-sync signal triggers signal sampling to determine the transmitted data symbol.*

QAM modems also have carrier-tracking and descrambler blocks. Newer high-speed QAM telephone modems need adaptive equalizers to correct phase and amplitude distortions in the transmission path.

The signal-to-levels block accepts digitized samples of the modulated signal and uses a quadrature downconverter to turn them into instantaneous ZQ values. To demodulator's performance can be gauged from ZQ plots which display ZQ values on a xy graph.

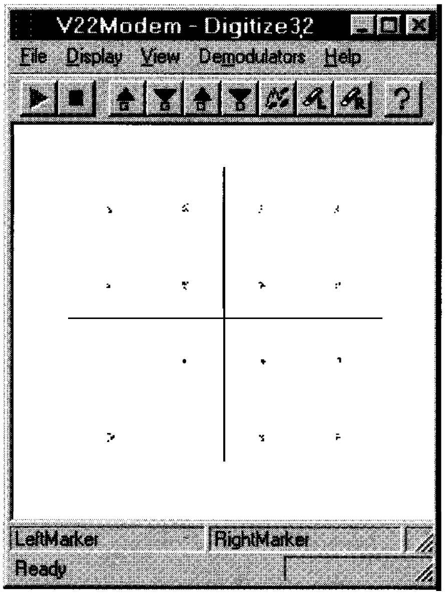


Photo 1-A *typical V22Modem-Digitize32 window displays* **the** *IQ plot for* **a** *V.22bis* **X-point** *QAM signal. V22Modem enables you to* **demodulate** *and* **display data** *from live* **V.22 and** *V.22bis modem signals or* **sampled .** *WAV* **files.**

The bit sync determines when to sample the instantaneous ZQ values to make the best estimate of the transmitted bits. And, the descrambler outputs the correct data bits.

A DEMODULATOR IN Ctt

We designed our code to be modular, work with data from several digitizers, run on different platforms and compilers, and take higher speed signals.The CV22Demod and CV22Bis - Demod classes accept single samples scaled from -1 .0 to + 1 .O and output the demodulated bits once per symbol.

The class is constructed with specific signal and demodulator information such as sample and symbol rates, bandwidth, gains, and feedback-loop coefficients. CV22Demod outputs an integer value equal to the dibit or quadbit, the string value of the dibit or quadbit, or the ASCII value of the transmitted character.

It's handy to have all the demodulator code in one class that can be called by a real-time or file-based routine. Listing 1 shows how to use the CV 22 - Demod class from a file-based routine.

SIGNAL TO LEVELS

The quadrature downconverter or mixer operates on each sample and converts the audio signal from a receiver or phone line into Z and Q signals. The frequency mixer multiplies the current incoming sample by the carrier numerically controlled oscillator's (NCO) sine and cosine values. The low-pass filters filter out the high-frequency terms the mixers produce.

For testing, we used signals from a pair of standard two-wire modems, operating in the leased-line mode and connected directly to each other and the sound card's line input. For telephone and other two-wire modems, both the low- and high-band channels are present in the audio signal.

It isn't necessary to band-pass filter the V.22 signals before the demodulator. Mixing with the appropriate carrier places the undesired channel's signal outside the pass band of the mixer's low-pass filters.

By changing the carrier frequency, the identical DSP code demodulates either channel. However, not band-pass filtering requires a higher order LPF, which means more time-consuming multiplies per second.

We implemented both FIR and IIR filters. Linear phase response is important for QAM signals. Designs sometimes use FIR filters because of their better linear phase response and the ease of matching the LPF to the de sired transmission filter.

We first used fourth-order IIR filters since they provided high attenuation for fewer multiplies and were easier to implement than FIR filters. This choice originally meant faster real-time performance and less time writing code.

We determined the necessary degree of the filter by trying second-, fourth-, and sixth-order LPFs designed by using the Interactive Filter Design Web Page [1] and looking at the quality of the IQ plots. However, these predesigned filter coefficients were only valid at a single-sample rate.

Since we wanted to experiment with different sampling rates at run time, we designed ClstOrderLPF and C2ndOrderLPF low-pass-filter classes. These classes calculate the IIR-filter coefficients based on a specified sample rate and cutoff frequency at program run time (see Listing 2).

We used equations and procedures from Do-While Jones' "Digital Filter Alchemy" (*INK* 61) to design filter classes for fast real-time applications. We first cascaded two C2ndOrderLPF filters to make a fourth-order filter for each arm of the quadrature mixer (see Listing 3).

We used FIR filters to improve the demodulator's performance by designing a LPF that better matched the impulse response of the transmission filter and channel. Because FIR filters require many more multiplies than equivalent IIR filters, we oversampled I and Q signals and downsampled (decimated) the signals to reduce the time needed for the FIR filter.

V22Modem-Digitize32 samples the analog signal using the sound card at a standard rate of 11,025 Hz. Therefore, the *I* and Q signals are oversampled at over 18 samples per signal.

The follow-on equalizer only requires two samples per symbol. So, by decimating the *I* and Q signals, we saved significant processing time for improved real-time performance.

Typically, to eliminate aliasing, a LPF band limits a signal before decimation. The antialiasing filter operates on each sample at the higher sample rate and uses a lot of processing time.

But, a FIR filter can combine filter and decimate functions because its output only depends on the samples in the filter at the output time. Every sample is shifted into the filter, but output calculations are only performed on 2 of every 18 samples for an effective IQ sampling rate of 1200 Hz or 2 samples per symbol.

For oversampled signals, using a decimating FIR filter helps to increase system performance when using a FIR filter. If we didn't implement this function, the 39 tap FIR filters would take too many multiplies and adds to run in real time on a '486.

It took about the same time to calculate the IIR and decimating FIR filters. In the end, the overall demodulator design drove the choice of using an IIR or FIR filter.

The equalizer design in CV22BisDemod uses two samples per symbol for the symbol-to-bit, carrier-tracking, and bit-sync functions, so it uses the

Listing 1—*OnFileRead() is the CIQPlot member function associated with the* **V22 Demod** *from the file dialog box. OnFileRead() reads* **digitized data** *from a* **.***WAV file and shows how to use CV22Demod* **to implement the V.22 demodulator. The complete** CIQPlot::OnFileRead() is in *iqplot.cpp. For a* **real-time** *implementation of CV22Demod, the V22, PlotPoint, and BitSyncPoint pointers should be global and only created and deleted when the modem is started and stopped.*

```
void CIQPlot::OnFileRead()
{
  (read data buffer from data file)
  //construct an instance of the V22Demod class
  V22 = new CV22Demod(m_Sample_Rate,
    m_NCO_Carrier,              //V22 = 1200 or 2400 Hz
    m_Plot_LPF_Fc,              //needed for mixer low-pass filter
    m_Plot_LPF_Gain,            //needed for mixer low-pass filter
    m_Plot_Sigma,               //needed for mixer low-pass filter
    m_Symbol_Rate,              //V22 = 600 sps
    m_Loop_LPF_Fc,              //needed for carrier-tracking loop
    m_Freq_Loop_Gain,           //needed for carriertracking loop
    m_Phase_Loop_Gain);         //needed for carrier-tracking loop

  // main loop to process data in buffer
  for (x = m_Start_Position;
         x < bytes_read-m_Buffer_Size; x+ = m_Buffer_Size){

    for (y= 0: y < m_Buffer_Size; y++){
      //model the real-time buffers received from sound card
      //calculated once per sample
      double tempdata = (double) ((data[x+y] - 127)/input_gain);
      V22->Quadrature_Mixer( tempdata);
      //call CV22Demod's quadrature mixer function
      (fill a point array for plotting to screen)
      if (V22->BPF_BitSync(V22->cos_carrier, V22->sine_carrier)){
        //check if bit sync has fired
        //all code inside BPF_BitSync loop runs once per symbol

        scrambled_dibit = V22->Level_To_Bits(V22->cos_carrier,
                          V22->sine_carrier);//update NCO freq.
        V22->Dec_Directed_NCO(); //for carrier tracking
        unscrambled_dibit = V22->Scrambled_Bits_To_Bits();
        if (m_Output_File){     //if output to file is chosen

          ascii_int = V22->ASCII_Frame_Sync(unscrambled_dibit);
          if (ascii_int){
            (output the ascii character to screen or file)
          }
        } // end of m_Output_File
      }// end of m_Bit_Sync
     sample_counter++;          //sample number processed
    }// end of buffer read
    (option to display IO point array)
    (option to display bitsynced symbol assignment)
  }// end of data file
```

Listing *2-In ClstOrderLPF, one of the* **DSP** *building blocks for this project, the* **filter** *coefficients are calucated at run time based on a specified sample rate and* **cutoff** *frequency.*

```
ClstOrderLPF I-Filter = ClstOrder(11025.50.0, 1.0);
                                      //construct a 1 pole, 50-Hz LPF
temp = I_Filter.LPF(current_sample);//return a filtered value
```

Listing 3—*The* `CV22Demod::Quadrature_Mixer()` *member function implements the QAM mixer block. The actual mixing is done by the* `tempdata*cos(NCO_radians)` *and* `tempdata*sin(NCO_radians)` *code. The output of the mixer is then low-pass filtered.*

```
void CV22Demod::Quadrature_Mixer(double tempdata)
{
  //multiple by the NCO carriers
  NCO_radians = NCO_radians + tempNCO* two_pi_X_inv_m_Sample_Rate;
  I_Value  = I_2_600_LPF->LPF(I_600_LPF >LPF(tempdata*cos
                (NCO_radians)));
  Q-value = Q_2_600_LPF->LPF(Q_600_LPF->LPF(tempdata*sin
               (NCO_radians)));
```

FIR filters. The open-loop bit-sync design operates at a higher sample rate, so it uses the IIR filters. Both demodulator designs run at similar speeds.

The CBDecimate class implements the decimating FIR filters. It defines all decimation functions and three FIR low-pass filters-hamming, hanning, and raised cosine. It inputs the down-converted I and Q signals as a complex number.

The decimation interpolates between samples to allow for noninteger decimation rates. However, this requires two filter calculations per output.

CBDecimate has two main member functions to process the data. The first is CBDecimate::PutSample, which inputs a new complex sample. PutSample() tests and returns TRUE if it's time to calculate a new output sample. CBDecimate::GetSampleO is then called to calculate the filter's complex output.

CARRIER TRACKING

QAM signals require the **local** carrier's NCO to be phased locked to the transmitter's carrier. This requirement isn't straightforward because QAM modulation results in a suppressed carrier.

CV22Demod uses decision-directed feedback (DDF) to track the carrier. Whenever the bit sync fires, determining that the symbol should be sampled, DDF calculates the phase difference between the current symbol and the nearest constellation point. That phase difference becomes the input to the carrier-tracking loop.

The CV22Demod::Quadrature_Mixer() member function generates the *I* (cosine) and Q (sine) carriers. A frequency sets the phase change per sample. The NCO calculates the instantaneous phase {NCO_radians) for each sample by adding the phase change per sample (tempNCO*two_pi_X_inv_m_Sample_Rate) to the previous phase value or sum.

For a carrier frequency of 1200.0 Hz, the phase change is 0.6839 radians per sample. For 1201 .O Hz, the phase change is 0.6845 radians per sample. The standard sin() and cos() functions use the new instantaneous phase to calculate the I and Q carrier waveforms.

As shown in Listing 4, TempNCO, a data member of the V22Demod class, holds the desired carrier frequency. It is the feedback or input variable for this NCO. We use the NCO as part of the carrier-tracking phase-locked loop by changing the tempNC0 frequency once persymbolin CV22Demod::Dec_Directed_NCOO.

Part of designing real-time systems is discovering and optimizing critical time-consuming portions of code. While there are no hard and fast rules, a typical DSP project might contain over 90% high-level code and less than 10% hand-assembled code.

The NC 0 value is calculated once per sample and is a good place for future real-time optimization. The FSINCOS '487 assembly-language instruction calculates both sine and cosine values in one instruction. By converting the Quadrature_Mixer'sNCO code to assembly language, we can potentially increase the speed by 1.5 times.

Since the mixer's low-pass filters are called every sample, they're a good place to start optimizing code-especially since both the NC 0 and the low-pass filters have a high potential for reuse in future projects.

The NCO also shows that, when possible in CV22Demod, we precalculate constants or use the inverse of constants (e.g., two_pi*inv_m_Sample_Rate} and multiply instead of divide. On the '486, it takes five times longer to divide two floating-point numbers than to multiply them.

One disadvantage of using DDF for carrier tracking is that bitsync and carrier feedback loops are interdependent.

But, DDF's big advantage is that decision- and feedback-loop calculations take place once per symbol (i.e., 600 per second) instead of once per sample (i.e., 11,025 per second). The carrier-tracking loop is also independent of the signal sample rate.

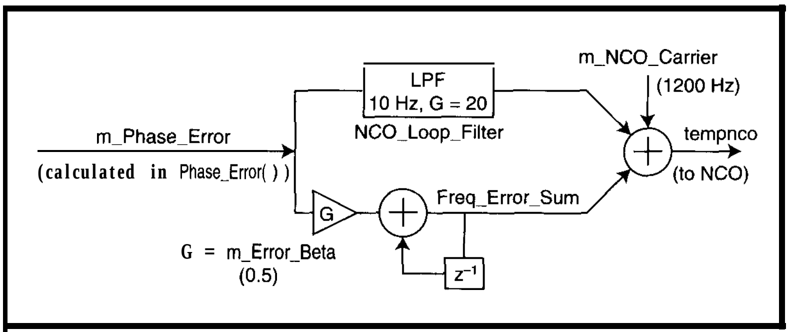
Figure 2 and Listing 4 show the carrier-tracking loop filter and the code. The loop filter is a standard proportional-integral (PI) design.

We determined the filter structure and coefficients by trial and error using sampled modem. WAV files. Freq_Error_Sum is the output of a integrator (pole at zero) and sets the DC loop gain. The one pole LPF filter, NCO_Loop_Filter, is an instance of the ClstOrderLPF class.

BIT SYNCS

There are many methods of bit or symbol synchronization. CV22Demod uses an open-loop bit sync, while CV22BisDemod has a closed-loop bit sync.

The open-loop design in Figure 3 uses band-pass filters and



'igure 2-A *standard proportional-integral (PI) feedback system tracks the transmitted carrier.*

mixers to regenerate the symbol clock and provide the bit-sync signal. We used an interactive Web page to calculate the resonating, high-Q IIR band-pass filter coefficients used by the `C2ndOrderBPF()` class [1].

Once again, the C++ class concept let us build one band-pass-filter class that is used three times in the bit sync. We chose high-Q IIR filters because they have narrow bandwidths with only a few multiplies and adds. The output of the bit sync's final 600-Hz filter is a sine wave.

The trick to using this open-loop QAM bit sync is finding where in the 600-Hz BPF filter's sine-wave output the bit sync should fire to sample the *IQ* signal.

Before writing the bit-sync function class, we modeled it with a spreadsheet. Since zero crossings are easy to detect, we used the positive-going zero crossing as a reference and determined the optimum delay time during modeling.

A disadvantage of this BPF technique for the bit sync is that it requires three filters and two squaring circuits per sample for a total of 17 multiplies and 12 adds each sample. At F_s = 11,025 kHz, there are 18.3 samples per symbol—significantly more than necessary.

Similar to the decimating FIR filters, if you think of the *I* and Q low-pass filters as 600-Hz antialiasing filters, you can discard three out of four samples and still have the signal oversampled by 4.5 times.

Eliminating every three samples saves four times the number of multiplies and adds, increasing the overall demodulator speed by 1.5 times. We interpolated the zero crossing and I and Q values for the bit sync to work with decimated samples.

The closed-loop bit-sync design uses the asymmetry of equalizer taps as the error signal to adjust the demodulator's symbol clock. As the symbol clock slowly drifts out of phase with the signal, the center of the coefficients slowly slides to left (early) or right (late] within the equalizer.

LEVELS TO BITS

The `CV22Demod::Level_To_Bits()` function uses a series of If statements to determine which symbol is closest to the current I and Q

Listing 4-The *LPF is* an instance of the Cl `stOrderLPF` *class.* `TempNCO` *is measuredin* radians. The two gain *terms* and *fhe LPF cutoff frequency can be modified in fhe* `V22modem32.ini` *file for* experimenting *with* **different** *loop coefficients.*

```
double CV22Demod::Dec_Directed_NCO()
{
  //Update the local oscillator's frequency
  Freq_Error_Sum = Freq_Error_Sum + m_Error_Beta*m_Phase_Error;
  // perfect integrator
  tempNCO = m_NCO_Carrier + NCO_Loop_Filter->LPF(m_Phase_Error) +
                Freq_Error_Sum;
  return tempNCO;
}
```

values and to assign the proper bits to the transmitted symbol.

Two If statements determine the quadrant change for V.22 and the second dibit of V.22bis. V.22 levels to bits can also be implemented with a series of Ifs to determine the delta phrase between the current I and Q values and the previous symbol's phase angle.

V.22 AND V.22BIS DESCRAMBLERS

The descrambler is the last step in producing output bits in the demodulator. V.22 and V.22bis share the same 17-stage scrambler and descrambler.

The descrambler is a tapped shift register that stores the last 17 bits [see Figure 4). The descrambler MOD2 adds transmitted bits 17, 14, and current together to form the unscrambled bit.

We implemented the descrambler using `CBRegister`, a shift-register C++ template class that stores data in a circular buffer. It's a fast and efficient way to access a series of values that need to be shifted one register per sample, as in FIR filters, time averagers, or delay lines.

The descrambler is implemented in theprivate `CV22Demod::V22_Scrambled_Bit_to_Bit()` member function which takes a scrambled input bit and returns a descrambled bit (see Listing 6). The `CV22Demod::Scrambled_Bits_To_Bits()` member function is customized for V.22 and V.22bis dibits and quadbits. `Scrambled_Bits_To_Bits()` is calledafter each symbol is demodulated. It in turn calls `Scrambled_Bit_To_Bit()` one bit at a time.

ADAPTIVE EQUALIZER

The equalizer adjusts or adapts the phase and frequency response of a FIR filter's coefficients. It's an attempt to correct for distortions in the communications channel and produce the best estimate of the current symbol.

An adaptive equalizer constantly adjusts coefficients, while a fixed equalizer only adjusts them once per training sequence. The V.22 and V.22bis specs don't require equalizers, but higher speed modems do.

Bandwidth-limited transmission channels are unable to transmit perfect (i.e., infinite) symbol pulses and smear past, current, and future symbols, causing intersymbol interference (ISI).

By using a proper transmission filter with zero energy at integer multiplies of the symbol time, the contributions of the smeared pulses are zero. However, distortion in the channel's phase and frequency response and truncated filters cause improper pulse alignment,

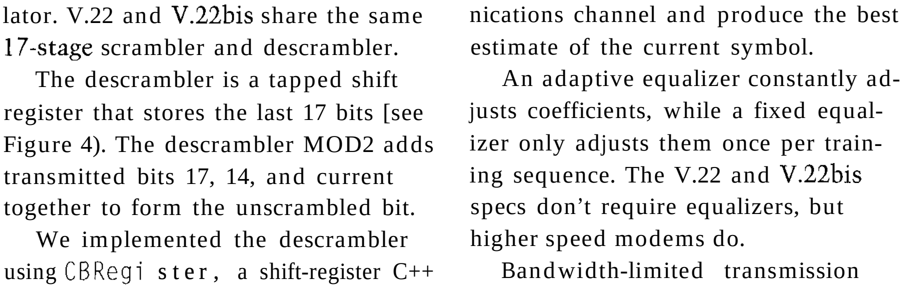


Figure *3-This open-loop QAM bit-sync design using three instances of the* `C2ndOrderBPF` *class with filter coefficients was designed using a Web page [1]. The design can be adapted to different symbol rates by seffing the summing BPF equal to the symbol rate and the input filters to one ha/f the symbol rate. See Listing 1 and* `CV22Demod::BPF_Bit_Sync()` *for code to implement the bif sync.*

Listing 5—*The* `BPF_BitSync()` *QAM bit synchronizer decimates three out of four output samples from the* `Quadrature` *Mixer and calculates the optimum time to sample the IQ values. See* `cv22.cpp` *for the complete listing.*

```
BPF CONSTRUCTORS:
//Fs=2765 Hz, Fc=300 Hz, Q = 100
I_300HzBPF = new C2ndOrderBPF
   (-0.9931839456,  1.5448876939,  293.4248888);
//Fs=2765 Hz, Fc=300 Hz, Q = 100
Q_300HzBPF = new C2ndOrderBPF
   (-0.9931839456,  1.5448876939,  293.4248888);
//Fs=2765 Hz. Fc=600 Hz. Q =1000
BitSync_600HzBPF = new C2ndOrderBPF
   (-0.9864142558,  0.4002915270,  147.2131352);

INT CV22Demod::BPF_BitSync
   (double I-input-value, double Q_input_value)
{
//routine to band-pass filter I and Q arms,
    then square each output (^2),
//add results and band-pass filter at symbol rate = 600 Hz
  (calculate filter values every fourth sample)
 ...
  double I-filtered = I_300HzBPF->filterloop(I_input_value);
  double Q-filtered = Q_300HzBPF->filterloop(Q_input_value);
  double sum_bpf = I_filtered*I_filtered + Q_filtered*Q_filtered;
  double baud-filtered = BitSync_600HzBPF->filterloop_gain(sum_bpf);

  (remainder detects zero crossings and interpolates I and C! values)
}
```

creating additional IS1 [2]. The equalizer tries to adjust its response to minimize ISI.

We implemented a two-sample-per-symbol complex linear adaptive-equalizer using a 17-tap complex FIR filter in `CBEqualizel_2V22`. `CBEqualizel_2` also uses the equalizer for the levels-to-bits and bit-sync functions.

A nine-symbol equalizer (17 taps] worked well. The center tap C_0 contains mostly energy from the current symbol and some from the past and future symbols C_{-k} to C_k.

The equalizer adds the current-symbol center tap with scaledversions of the past and future symbols. With the correct symbol clock and tap coefficients, the energy from past and future symbols can be subtracted out, reducing the ISI. Figure 5 shows how an equalizer improves the SNR of a V.22 signal.

Our complex linear adaptive equalizer (see Figure 6) uses the minimum mean squared error to calculate the new tap coefficients. The equalizer converges to the coefficients producing the smallest overall mean squared error from the expected constellation points.

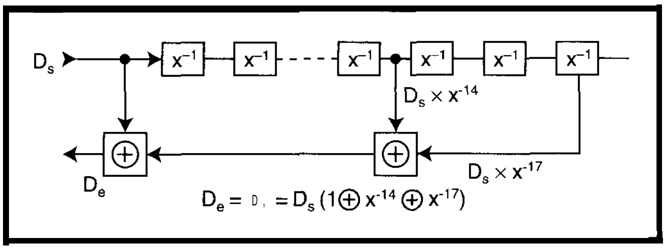


Figure **4**—*V.22 and V.22bis modems use a shift register to descramble the transmitted bit.*

A complex class, `CBCompl ex`, simplifies the mathematics. Each storage register in the filter contains the *I* and Q values as a complex number.

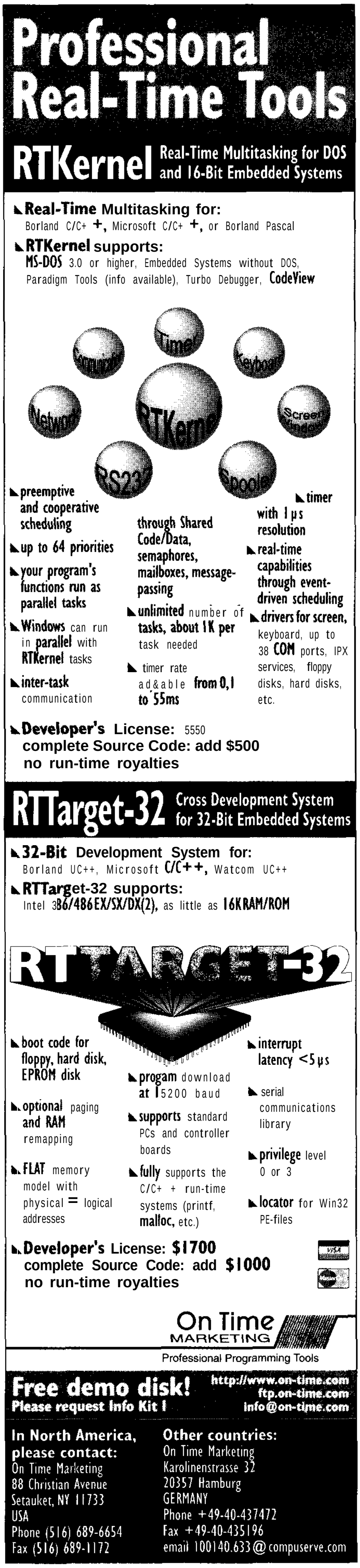
The equalizer algorithm starts with C_0 set to 1 and all other coefficients to 0. Once per symbol, the equalizer calculates the output symbol from the sum of the coefficients times the tap values.

The output symbol is subtracted from the closest point in the constellation (DDF). This produces a complex phase-error vector that is scaled by a small value (0.2).

The scaled error is multiplied by each tapped value in the filter and added to the coefficients. The complex phase error is also used as the feedback signal for the NC 0 carrier-tracking loop.

FRAME SYNC

The frame synchronizer recognizes a valid frame marker and groups the demodulated bits into data blocks.



Data blocks and frame markers are defined as part of the communications protocol (e.g., ASCII or TCP/IP) and are not part of the modem's specifications.

The CASCII class (found in cascii.h and cascii.cpp) contains the ASCII frame-sync code to detect the start of the ASCII character or data block and reverses the bits since ASCII is transmitted LSB first. ASCII_Frame_Sync() returns an integer equal to the ASCII character value of the data block.

For ASCII asynchronous data, the mark condition or no transmitted data is al, the same as a stop bit. The pattern 10 occurs if two data blocks are sent immediately after one another or if there is a period of no transmitted data followed by a valid data block. ASCII_Frame_Sync searches for a stop bit (1) and start bit (0) followed by eight data bits.

The CV22Demod function is completed after the ASCII value is returned. The program has several options to display the demodulated bits or ASCII data to the screen or a file for testing and analysis like a simple data analyzer.

Listing 6—*Scrambled_Bit_To_Bit() implements the* **V.22** *and V.22bis 17-bit-long descrambler from the V.22 CCITT specs. It demonstrates how to use the CBRegister template class to store the previous 17 scrambled bits.*

```
From cv22.h header file
  CBRegister<int> data:    //defines data as integer register

From cv22.cpp
int  CV22Demod::Scrambled_Bit_To_Bit(int scrambled_bit){
  int temp17 = data[16];   //seventeenth storage location
  int temp14 = data[13];   //fourteenth storage location
  int unscrambled-bit = temp17^temp14^scrambled_bit;
  data.Put(scrambled_bit); //add scrambled bit to register
  return unscrambled-bit;
}
```

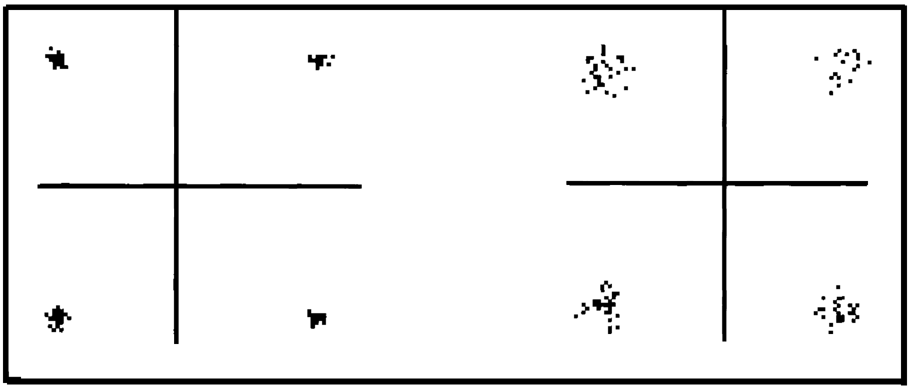


Figure 5—*IQ plots show the SNR improvement using the CBEqualize1_2 adaptive-equalizer class. The last 400 symbols are plotted. Adaptive equalizers allow for denser symbol constellations and thus higher data rates.*

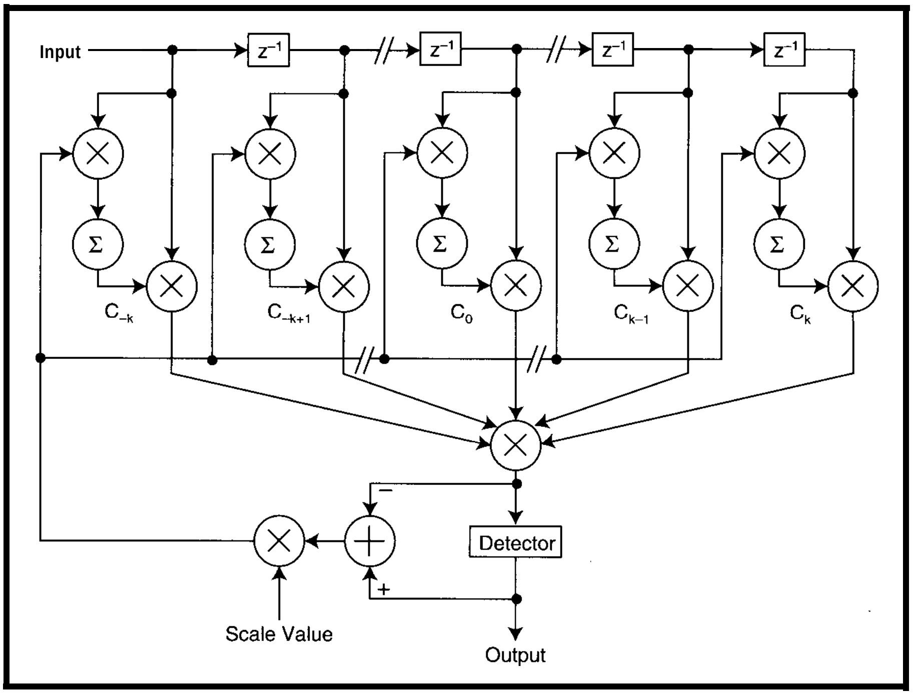


Figure 6—*This diagram shows the complex linear adaptive equalizer used by* CV22BisDemod. *The code for the* CBEqualizel_2V22() *class is in* BEqualizel_2.cpp *and* BEqualize.h.

Watching the ASCII characters gives feedback about the demodulated bits. For embedded systems, add a custom frame marker, data-block detector, data-logging capability, or other data-processing functions to the demodulated bits.

WHERE TO GO FROM HERE

We aren't done optimizing the real-time code. A few routines could benefit from some assembly language.

We'll next experiment with higher speed QAM modems since C V 2 2 Demod is almost already a generic QAM demodulator. The idea of real-time fax demodulation is intriguing.

We hope you've gained some ideas on how you might use DSP in your own embedded controllers and communications projects. ❑

Matt Park holds a BSEE from The Ohio State University and an MSEE from The Johns Hopkins University. He is an engineering consultant for communications and electro-optics. You may reach Matt at mattpark@transend.com.tw.

Brian McLeod holds a BSEE from Akron University and is an engineering consultant for communications and signal processing. You may reach him at bnmcleo@transend.com.tw.

REFERENCES

[1] G. Fisher, "Interactive Filter Design Web Page," <http://dcpul.cs.york.ac.uk:6666/fisher/mkfilter/>.

[2] J.G. Proakis, *Digital Communications,* McGraw-Hill, New York, NY, 1983.

SOFTWARE

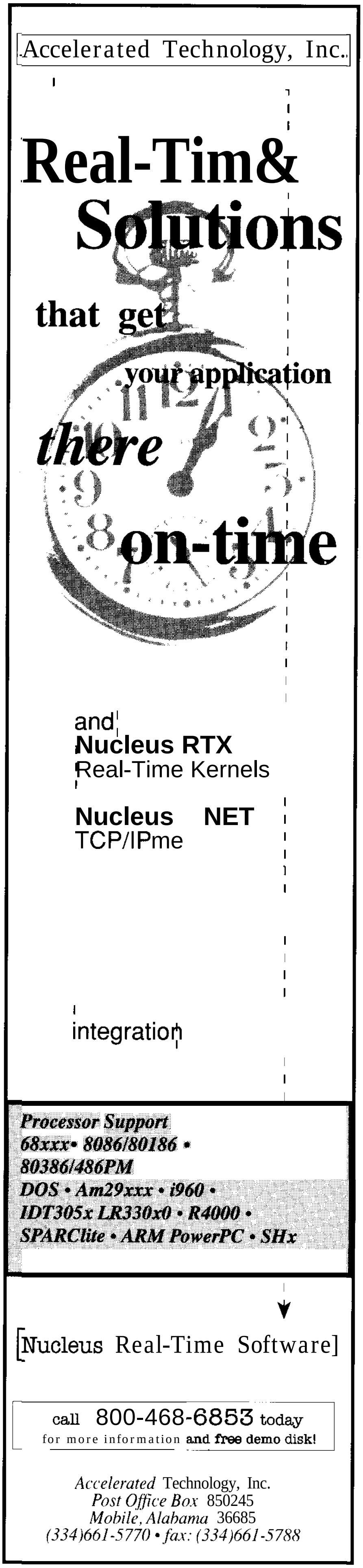
Software from this article can be downloaded from the Circuit Cellar BBS, the Circuit Cellar Web site, and from Software on Disk for this issue. Please see the end of ConnecTime for downloading and ordering information.

SOURCES

Visual C++ 4.0, Visual C++ 1.52, Windows 95, Windows **NT**
Microsoft Corp.
One Microsoft Way
Redmond, WA 98052-6399
(206) 882-8080
Fax: (503) 244-726 1

I R S

404 Very Useful
405 Moderately Useful
406 Not Useful



FEATURE ARTICLE

Frank Gao

Algorithms and Software for V.34 Modems

Software modems are more flexible and cost-effective than nonprogrammable chips. Franks shows us the operational principles and software building blocks of fax/modems so we can take advantage of this edge.

imerest in software implementations of V.34 is rapidly increasing due to the flexibility and attractive cost-performance ratios software modems offer.

Traditionally, implementations of modem and fax data pumps are burned into dedicated chip sets. Engineers using these off-the-shelf, nonprogrammable chips can usually ignore their internal operational mechanisms.

But, with the increasing popularity of the software implementations of modem and fax, designers need to understand their operational principles and software building blocks.

In this article, I present the major signal-processing functions a V.34 data pump requires and its software implementation. The principles are common to other fax/modem techniques, but I use V.34 software from GAO Research and Consulting as an example.

V.34 MODEM HANDSHAKING

The V.34 modem data-pump software is composed of three parts-the handshaking, the transmitter, and the receiver.

The V.34 modem startup has four phases. Phase 1 is defined in ITU V.8, a universal automode scheme for modems, faxes, and other communications devices. Phase 2 sets the modem parameters and performs line probing and ranging. Phase 3 is used for initial training of the equalizer and echo canceler, and Phase 4 offers the final training and exchange of data-mode modulation parameters.

V.34 MODEM TRANSMITTER

The transmitter's structure (see Figure 1) follows the ITU V.34 standard. It has five parts-the scrambler, framer, encoder, filter, and modulator.

The scrambler randomizes the input binary data to spread the transmit-signal spectrum over the transmit band. It facilitates effective transmission of data over the telephone channel and improves the convergence of the adaptive equalization and echo cancellation in the receiver.

The framer includes a superframe, a data frame, and a mapping frame. The superframe consists of seven or eight data frames for different symbol rates.

The data frame has 12-16 mapping frames for different symbol rates. The mapping frame has four 4D symbol intervals, each consisting of two 2D symbol intervals.

The encoder consists of a shell mapper, a differential encoder, a precoder, a Trellis encoder, and a nonlinear encoder. Three new techniques are employed in the encoder-multidimensional Trellis-coded modulation (MD-TCM), shell mapper, and precoding equalization.

MD-TCM combines coding and modulation so the modem benefits from high reliability with limited transmission frequency bandwidth. The standard recommends three convolutional encoder structures for the transmitter. The three Trellis codes—16-state rate-2/3, 32-state rate-3/4, and 64-state rate-4/5, all with 4D QAM constellation-can achieve asymptotic coding gains of -5 dB.

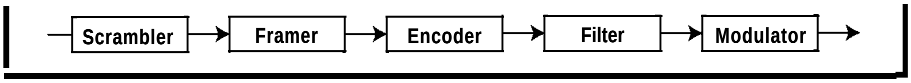


Figure I--The *transmitter of the* **V.34 modem processes the data to be transmitted** *to* **minimize the effects of** *channel* **noises and** *to* **improve the** *convergence* **of the adaptive equalization** *and echo cancellation* **in** *the* **receiver.**

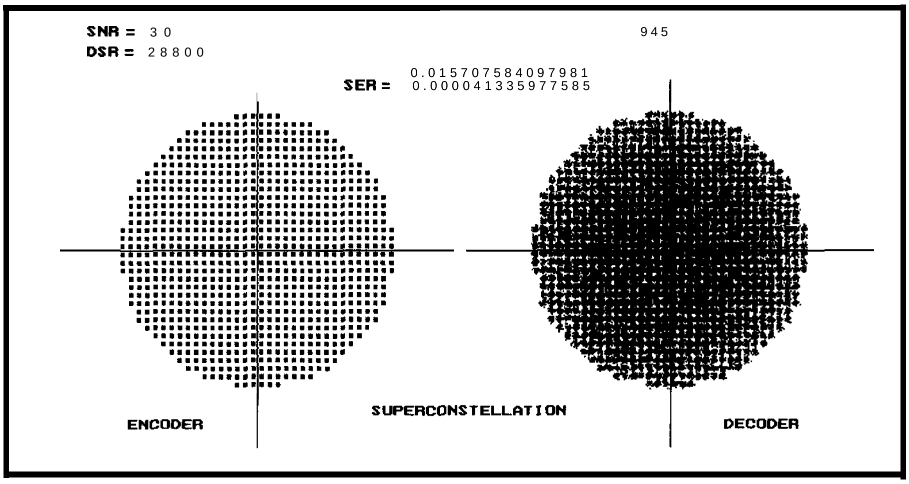


Figure 2—*The constellation diagram on the left is from the transmitter of a V.34 modem. The one on the right is from the receiver.*

Shell mapping reduces the average signal energy and improves modem signal-to-nce with the same precoding. The technique, a nonlinear equalization scheme, reduces the effects of noise enhancement in the equalizer.

The pulse-shaping low-pass filter in Figure 1, called a raised cosine filter, eliminates intersymbol interference (ISI) on the band-limited PSTN.

Unlike the ideal pulse-shaping filter, the raised cosine filter is realizable and has tails of decay proportional to $1/t^3$. So, the timing errors in the modem have a much less dramatic effect on the amount of ISI in the raised cosine filter. The pulse-shaping filter usually spans four baud intervals.

The ITU V.34 modem uses quadrature amplitude modulation (QAM) in the transmitter and receiver. The encoded digital sequence is amplitude modulated and converted to analog signal for transmission.

The double-sideband QAM in the V.34 modem is an efficient modulation technique in terms of bandwidth. In QAM, two quadrature carriers are modulated by two separate information-bearing signals-the real and imaginary parts of QAM. The V.34 modem signal set was designed to give enough distance between members of each subset.

Figure 2 shows the signal constellation for the V.34 modem. Signal constellations consist of complex, valued signal points.

The point with the smallest magnitude is labeled 0, the point with the next larger magnitude is 1, and so on. When two or more points have the same magnitude, the point with the greatest imaginary component is first.

V.34 MODEM RECEIVER

The ITU V.34 standard doesn't specify the receiver's structure, so the implementation is more flexible.

As shown in Figure 3, the receiver is a more complicated component. Several functions are the inverse of the transmit functions. The receiver consists of seven major functional blocks.

The AGC operates over a wide dynamic range and maintains the output signal at a constant level necessary for the proper operation of the receiver algorithms. In the receiver, several modules use amplitude thresholds to make their decisions.

Receive signals are typically adjusted by two AGC stages. The first signal is set during training and retraining, and the second is dynamically adjusted.

The samples received by the modem suffer from IS1 which results from the linear amplitude and phase dispersion in the channel. These broaden the transmitted signals and cause them to interfere with one another.

To increase transmission performance, a filter estimates the channel's inverse transfer function and performs channel equalization. Usually, a linear adaptive transversal filter is used. Its coefficients are adjusted by the Least Mean Square (LMS) algorithm.

The purpose of timing recovery is to recover a clock at the symbol rate or a multiple of the symbol rate from the modulated waveform (see Figure 4). This clock converts the received continuous-time signal into a discrete-time sequence of data symbols.

Timing recovery imposes requirements on the modulation technique not present when a separate clock is available. The strength of the timing information in a signal is affected by the statistics of the signal, line code, and pulse shape.

In pass-band systems, the carrier frequency is generated by a timing reference in a transmitter. Coherent demodulation of a pass-band signal requires the same carrier frequency and phase to perform the demodulation.

If symbol timing is known, carrier frequency can be derived. Symbol timing can be derived without knowing the carrier phase. When a receiver starts receiving data, it should derive the timing before estimating the carrier phase and adapting the equalizer.

The hybrids in a typical telephone system cause near- and far-end echoes, which are harmful for data

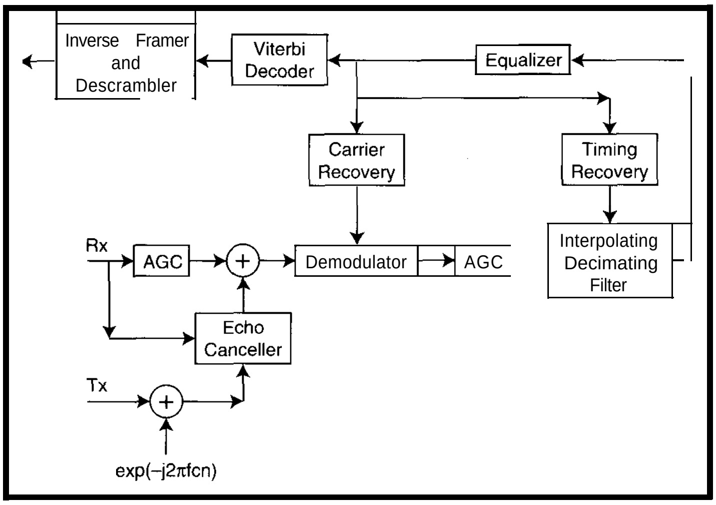


Figure 3—*The receiver of a V.34 modem performs such functions as demodulation, decoding, echo cancellation, equalization, and carrier and timing recovery.*

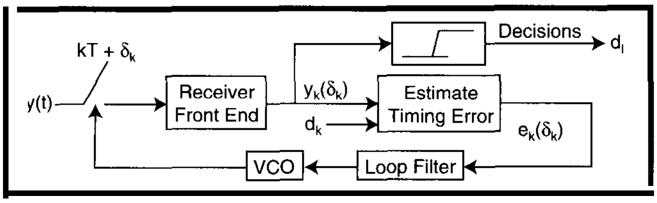


Figure 4—*Timing recovery recovers a clock at the symbol rate or a multiple of the symbol rate from the received data.*

transmissions over telephone networks.

The far-end echo canceler is weaker than the near-end echo. However, the far echo may be frequency translated when it passes through frequency converters (i.e., modulators and demodulators) on the four-wire circuits.

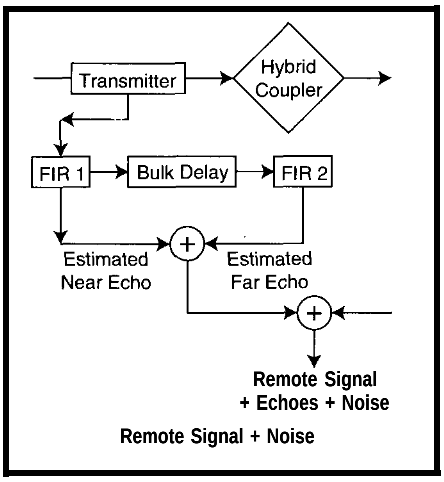
This phenomenon, called phase roll or frequency offset of the far echo, is a nonlinear impairment. So, nonlinear compensation techniques must be used for far-echo cancellation.

Near- and far-end cancelers eliminate the echoes. The echo cancelers use the received and delayed transmitted signals to estimate the echoes. They cancel them by subtracting the estimated echo signals from the received signals as shown in Figure 5.

To model the echo response precisely, the echo canceler must be trained using adaptive algorithms. The LMS algorithm is most widely used because of its computational efficiency.

The Viterbi decoder is a maximum-likelihood decoder used in the V.34 modem. It chooses the symbol sequence most like the transmitted signal sequence via some metric (e.g., the minimum Euclidean distance).

The symbol sequences in the V.34 modem consist of 4D symbols. One 4D symbol is the concatenation of two consecutive 2D symbols which are points in the QAM constellation.



gure 5—*The* echo canceler consists of *Rear-* and *far-*
ıd cancelers.

V.34 MODEM IMPLEMENTATION

The increased interest in the V.34 modem technology has been partially due to such new applications as the DSVD modem and H.324 video phone. GAO has developed V.34, DSVD, and H.324 software for DSP- or microprocessor-based soft modems.

The software has three versions. One version, coded in ANSI C, is for Pentiums, microprocessors, and floating-point DSPs. The two others are coded in the DSP assembly languages of Texas Instruments' TMS320C54*x* and Analog Devices' ADSP-21*xx*.

SOFTWARE-MODEM MARKET

V.34 modems have been driving the modem market since ITU released the standard in 1994. This should continue another six or seven years.

Software implementations of V.34 offer attractive cost-performance ratios and flexibility. I believe they'll soon play a more significant role than modems with fixed-function IC chips. ❑

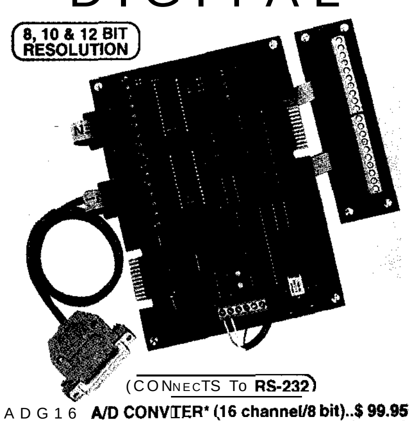
Frank Gao is the founder and CEO of GAO Research and Consulting, a leading supplier of software-modem technologies. You may reach Frank at gao@io.org.

SOURCES

V.34 **modem and software**
GAO Research and Consulting, Ltd.
55 Nugget Ave., Ste. 204
Scarborough, ON
Canada M1S3L1
(416) 292-0038
Fax: (416) 292-2364

I R S

407 Very Useful
408 Moderately Useful
409 Not Useful



Empowered Digital Filtering

FEATURE ARTICLE

Brian Senese

Due to the memory and processing power restrictions within an embedded system, it is even more important to pick the best filter for the job. Brian provides insight into which filter to pick for specific applications.

Oilter design using DSP techniques is very simple if you can calculate the coefficients necessary for its implementation. I find it frustrating, however, after reading about digital filters, to discover that the most critical information (i.e., the filter coefficients) isn't readily available.

In this article, I'll discuss filter design and provide you with a PC-based software tool that calculates coefficients. The many facets of digital filtering should be treated as simply as they're presented. General filter derivation, sampling theory, and coefficient quantization are topics best left for your own investigation.

BASIC SAMPLING

To begin, let's explore a basic sampling system. A typical configuration consists of front-end preparation, signal processing, and back-end signal reconstruction.

Preprocessing the analog signal involves passing signal energy through an antialias filter, which removes frequency components above the highest frequency the digital filter can handle.

Next, a sample-and-hold function makes a copy of the signal at a particular instant in time and holds this sample, so it provides a steady signal to the ADC.

The ADC then transforms the analog signal into its digital counterpart, so it's ready for processing. The rate at which analog signals are sampled is often governed by a timer, ensuring that samples are taken at constant and periodic intervals.

The processor manipulates this signal, as in the case of filtering, and writes the result to a DAC, which generates an associated analog value. Because digital values are written at periodic intervals, the analog output of the DAC appears stepped.

A reconstruction filter smooths the step-like response. That is, all high-frequency components created as a result of the D/A transformation are removed.

The maximum frequency acceptable for processing by the filter is linked to the sampling rate and defined as:

$$\text{Max frequency} = \frac{\text{Sampling rate}}{2}$$

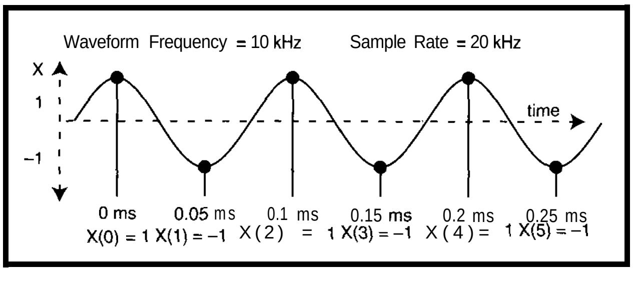
Removing analog-signal energy above the maximum frequency prevents aliasing from corrupting the desired filtered result. Sampled systems are characterized by this effect.

Signal energy appearing above the maximum frequency is reflected back into the frequency band of interest. The analog signal is captured through a sample-and-hold process that freezes the analog-signal level, allowing an accurate conversion by the ADC from its analog to digital representation.

The digital word's size determines many things-most importantly, the signal-to-noise ratio (SNR) of the system. For example, 16-bit representations take longer to convert from the analog source and are often more expensive to implement.

However, they yield superior results. As a general rule, 6 dB of SNR is

Figure 1—*A 10-kHz waveform is sampled at a 20-kHz sampling rate, capturing all relevant frequency information.*



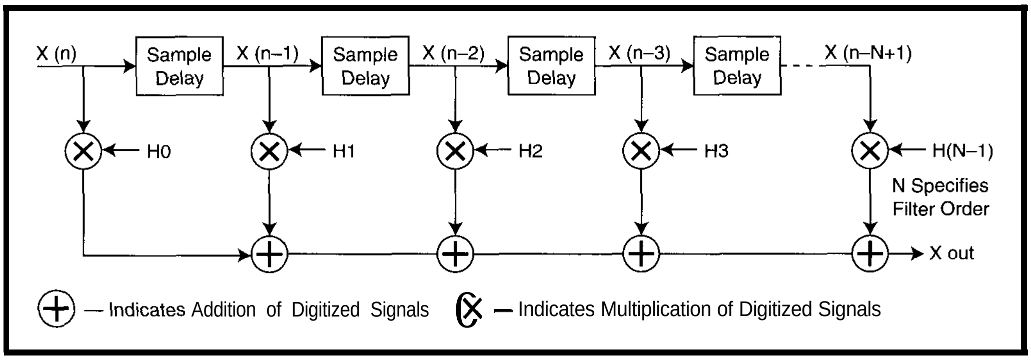


Figure 2-A *FIR filter* **is represented** *with H0,H1,H2,* **and so on being** *the filter* **coefficients and X(n),** *X(n-1), X(n-2),* **and so on being** *the* **ADC data samples.**

gained for every bit in the sample. That is, 16 bits of unsigned data defines a system capable of achieving a 96-dB SNR.

The processor reads and stores this sample, using it to calculate the next filtered value to be written to the DAC for signal reconstruction. Several input samples taken at consecutive time intervals are required to generate a single output value.

Reconstruction of an analog signal-the final step-is accomplished by writing the calculated digital-value stream to a DAC at a sample rate identical to that used in capturing the original analog waveform. A low-pass filter inserted at the output of the DAC removes unwanted harmonics generated by the DAC's step action.

A 10-kHz sinusoidal waveform requires the sample rate to be at least 20 kHz (see Figure **1**). All signal energy from 0 Hz to 10 kHz is also captured.

Signals appearing above **10** kHz alias back into the band of interest and cause distortion. But, an antialiasing filter prevents this.

So, in a basic sampling system, you should remember to:

- sample at a rate slightly greater than twice the maximum frequency you wish to process (this gives you a safety margin since digital filtering is an imperfect process)
- filter the incoming analog signal and eliminate any energy above the maximum frequency as defined earlier
- filter the output of the DAC to remove generated harmonics

FILTER CONCEPTS

There are two basic filter types—Finite Impulse Response (FIR) and Infinite Impulse Response (IIR).

FIR filters are labeled as finite because an output is generated only as long as a nonzero sample is present in the filter. They are the easiest filters to design and are generally used in communications-specific applications. In a modem design, for example, the filter output displays a desirable linear phase response over frequency.

FIR filters require more coefficients than IIR filters to realize the same filter-response characteristic. More processing cycles and memory are needed, making the implementation of FIR filters sometimes impractical—especially for microcontrollers.

IIR filters use feedback to generate an output. They have an infinite response since energy is always fed back into the filter. They require fewer taps (or stored signal samples) **to** realize the same response as an FIR filter.

One disadvantage of an IIR filter is in its group-delay characteristic. The output-signal phase response is nonlinear over the frequency band, and this characteristic is undesirable for communication systems.

IIR filters are an excellent choice for data-acquisition systems and are best suited for embedded-microcontroller systems. The PIC application handbook outlines an IIR design for use on PICs equipped with ADCs.

Listing I--This **routine is a** realization of a *FIR filter of* order 13. *All coefficients* are set *up* prior *to* **execution and are** extracted from the filter-design *software.* Note *that* **coefficient accuracy is very** *important* **for your** *filter to* **respond** *well to the* **original specification. The** *data* **value read** *from the ADC* **is assigned** *to* X(fi 1 ter_order - 1) *and the output value is* X_out.

```
/* placeholders for sampled data, filter coefficients, and
   final value */
double X[14], H[14], X_out;
int filter-order;

filter-order = 13;                         /* define filter order */

/* set coefficients for the filter as provided by the software */
H[0]  =  .0074656;
H[1]  = 0.0;
H[2]  =  .0377190;
H[3]  =  .04292241
H[4]  =  .0796275;
H[5]  =  .290973;
H[6]  =  .4;
H[7]  =  .290973;
H[8]  =  .0796275;
H[9]  =  .04292241;
H[10] =  .0377190;
H[11] = 0.0;
H[12] =  .0074656;

/* Run this code segment after reading ADC value. Each value (and
   preceding value) is used to calculate the filter output. All
   data values are shifted before next calculation. */
/*data input value is digitized sample read from ADC */
X[filter_order-1] = data input value;

X_out = 0.0;                               /*clear calculated value */

for (j = 0; j < filter-order;  j++)  /*calculate filter output */
   X_ out += H[ j ] * X[ filter-order 1 - j);
/*shift all data sample values in the filter before generating next
  filter outout value*/
for (j = 0: 'j < filter-order  - 1; j++)
   X[ j ] = X[j + 11;
```

Filters are defined mathematically and use a series of past data-sample values with a set of filter coefficients. Through a series of multiplications and additions, a singular, filtered output value is generated each time a new input is read. All processing must be completed before the next data sample is made available from the ADC.

The algorithm is interrupt driven, with the interrupt source being a clock that determines the sample rate. On invoking the routine, the previously calculated data point is written to the output DAC.

A new data sample is read from the ADC, calculations are made, and a new output value is generated. This value is held in memory until the next interrupt cycle.

The routine then shifts all sampled data in preparation for the new data value to be read in on the next cycle. Following that, the routine waits for the next interrupt.

Variations on this theme are possible. Other processes or instructions can be executed in place of the wait state I use. The most important rule of the system is that the filter interrupt routine must be serviced immediately and allowed to complete before another filter interrupt is generated.

Table l--These parameters *define fhe* desired filter responses for *IIR and FIR filters.* Calculated *filter coefficients in* ***the*** *examples are* based on *these specifications.*

	FIR	IIR
Low-Pass Filter	Kaiser window	Butterworth
System Sample Rate	10 kHz	10 kHz
Passband Ripple	1 dB	2 dB
Stopband Attenuation	40 dB	30 dB
Passband	O-l 000 Hz	O-1000 Hz
Stopband	3-5 kHz[1]	4-5 kHz

[1] 5 kHz is half the sample rate.

FIR filters use previous knowledge of the sampled signal to generate a filtered version of the incoming signal. The FIR filter is depicted in Figure 2.

X(n) is the digitized value read from the ADC. X(*n* - 1) is the previously sampled value, X(*n* 2) is the value read two sample periods ago, and so on. The filter order (or length) determines the roll-off response. Larger filter orders provide better filter characteristics. In Figure 2, *N* defines the filter order and *n* identifies the sample just taken.

Imagine, for example, that the ADC presents the sample marked as X(5) from Figure 1. In this case, *n* is 5 and, for a filter order of 4 (i.e., *N* = 4), you require previous sample values X(4), X(3), and X(2). These values are used to compute the next single output value.

The next sample read by the processor would be X(6). All sample values inside the filter must shift to the right for the new calculation to be correct.

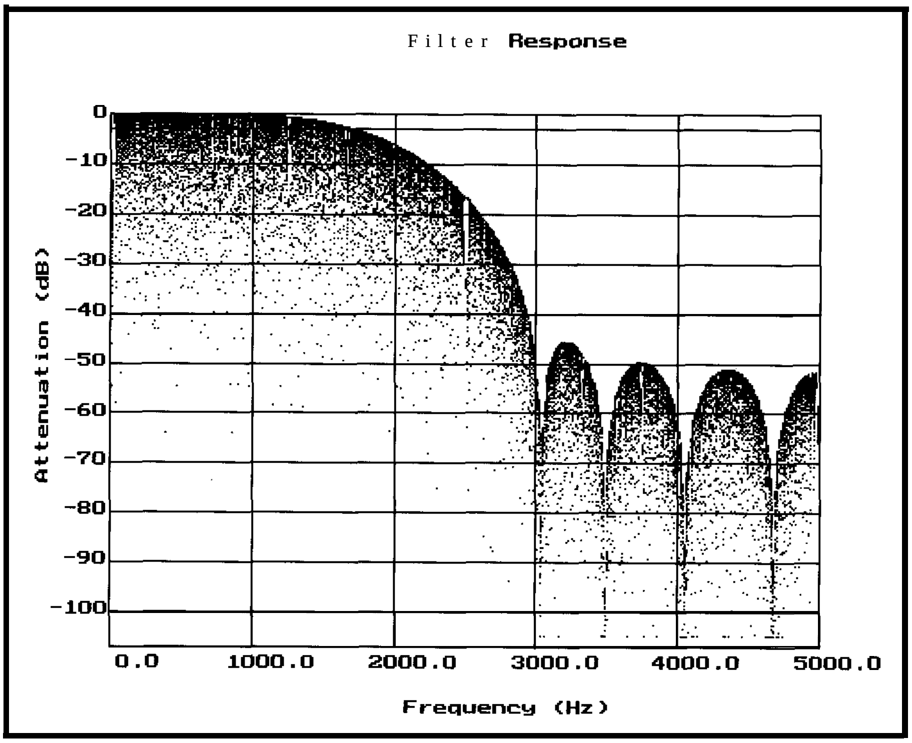


Figure 3—*This FIR filter response is generated by fhe filter-design* ***software*** *using fhe simulation mode of operation.*

Mathematically, the filter is expressed as:

$$\text{Signalout} = [H0 \times X(n)] + [H1 \times X(n-1)] + [H2 \times X(n-2)] \ldots [H(N-1) \times X(n-N+1)]$$

where *N* is the filter order and *n* represents the current sample value.

An FIR filter design starts by defining parameters that govern its performance.

For instance, you must know how much ripple is tolerable in the passband, the passband frequency (the frequency where the filter attenuates the signal by approximately 3 dB with respect to the passband), the stopband frequency, stopband attenuation, and the stopband ripple. The system sample rate must also be known.

After specifying all filter parameters, enter the values into a filter-design program. It first determines the filter order required to provide the desired response.

Then, it generates the coefficients for the filter. The order of the filter tells you how many taps to include in the algorithm. The filter coefficients are entered as constants.

The code segment in Listing 1 (written in C) realizes a FIR filter of order 13. Filter constraints for the design are shown in Table 1.

Entering these parameters into the digital-filter program yields a set of coefficients. These constants are assigned to an array which facilitates easy data manipulation.

Figure 3 illustrates the filter response of the designed FIR filter as presented by the digital filter-design program.

IIR FILTERS

Designing an IIR filter is very different from the FIR design process. An IIR filter is realized by cascading second-order IIR filter sections.

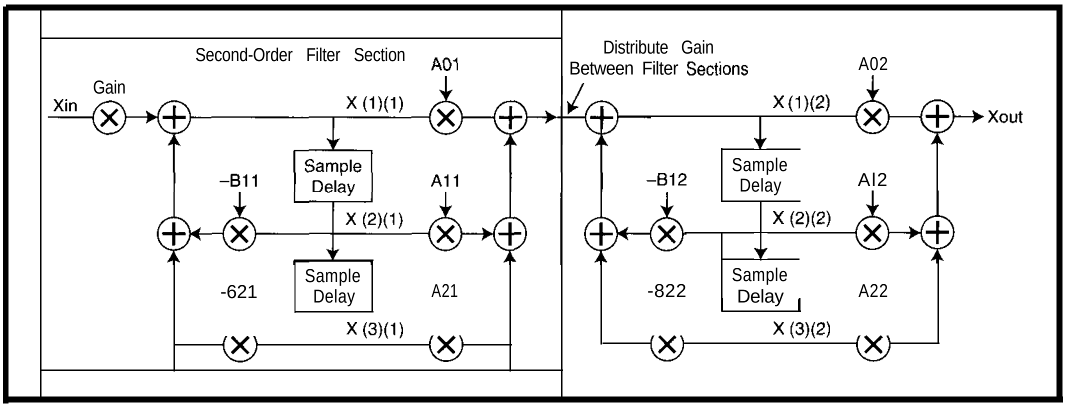


Figure 4-Several second-order *IIR filter sections can be* cascaded *to create* filters *of greater order which generate better filter-response* characteristics.

In Figure 4, a second-order filter section is identified. Filter coefficient values, defined as B1*, B2*, A0*, **Al** *, and A2*, are associated with specific second-order filter sections.

In Figure 4, the second identifying integer is linked to the filter-order section. This link becomes important when assigning coefficients to the filter structure because they're labeled **as** such in the filter-design software.

An IIR filter of a given order is simpler than a FIR filter of the same order, thanks to the use of feedback. Feedback applied to the incoming signal has the effect of reducing filter order.

In other words, it may only take a fourth-order IIR filter to yield the same filter response (in amplitude) as an eighth-order FIR filter.

Designing the IIR filter is almost identical to developing the FIR filter—only the algorithm changes. Mathematically, the second-order IIR filter section can be expressed as :

$$X(1)(1) = [Xin \times Gain] + [-B11 \times X(2)(1)] + [-B21 \times X(3)(1)]$$

$$Xout = X(1)(1) + [-A11 \times X(2)(1)] + [-A21 \times X(3)(1)]$$

Realizing an IIR filter with an order greater than 2 is accomplished by cascading sections together as illustrated in the fourth-order filter of Figure 4. The output of the first second-order section becomes the input to the next filter section.

Filters of even greater order are created by the continued addition of second-order sections. Note that Xin (the data sample value from the ADC) is not the same as the value entering the first data buffer (X(1)(1)).

A new coefficient, Gain, now appears in the equation and is required to scale the filter response. Because feedback is used by the filter, the input signal Xin must be reduced to prevent filter instability. It is recommended that in an actual filter design of order greater than 2, the Gain be spread across several sections equally.

If, for example, you've generated all the filter coefficients and have a Gain factor of 0.04, use a Gain of 0.2 for the first section and a Gain of 0.2 for the second section (providing an overall Gain of 0.04). Gain is only shown in one location in Figure 4 to prevent confusion since the filter-design program provides a single value.

The code segment realizing the second-order IIR filter is given in Listing 2. Filter constraints for the design are shown in Table 1.

The filter-response plot generated by the filter-design software displays a characteristic that adheres very closely to the specification.

ROLL YOUR OWN

You now know how to design digital filters without experiencing the pain of understanding the theory behind their operation. The benefits of this technology include being able to:

Listing 2-h *this example of a second-order IIR filter, the ADC value fed info the filter algorithm is assigned to X[0][1]. The calculated output value is X_out. Note that after all calculations are completed, the* data values *stored in* fhe *filter* **are** *shifted in preparation for the next calculation.*

```
/*placeholders for sampled data, filter coefficients and final value*/
double  X[4][2],B[3][2],A[3][2],X_out;
int filter_sections;

filter-sections = 1; /* specify one second-order filter section */

/* set coefficients for filter as orovided bv software */
B[1][1] = -1.0362927;
B[2][1] =   .3682664;
A[0][1]  =  0.5;
A[1][1]  =  1.0;
A[2][1]  =  0.5:
Gain     =   .165968;

/* Run code segment after reading ADC value. Each value is used
   (as are preceding data values) to calculate filter output. Data
   values are shifted before next calculation. The input value Xin
   in Figure 5 is labeled X[0][1] in following equation. */
/* data input value is digitized sample as read from ADC */
X[0][1] = data input value;

/* Calculate filter output */
for (j = 1;  j < filter-sections + 1;  j++) {
  X[1][j] = (Gain * X(0)(j)) - (B[1][j] * X[2][j]) - (B[2][j] *
             X[3][j]);
  X_out   = (X[1][j] * A[0][j]) + (X[2][j] * A[1][j]) + (X[3][j] *
             A[2][j]);
/* Shift data sample values in filter before generating next
   filter-output value. X[0][2] now contains output of filter
   and can feed the next filter section or DAC. */
   X[0][j+1] = X_out;
   X[3][j]   = X[2][j];
   X[2][j]   =  X[1][j];}
/* end of FOR statement */
```

- remove noise from an incoming signal without expensive hardware
- adjust the filter response characteristics in real time by simply changing filter coefficients on-the-fly
- filter a frequency to eliminate an unwanted signal [e.g., 60-Hz noise)

Enhancing your data-acquisition system or embedded-controller application through the use of digital filtering is now within your grasp. Consider yourself empowered. ❑

Brian Senese has developed wireless data-communication systems at Bell-Northern Research and PCSI for the last eight years. DSP is one of many technologies that hold his interest. He may be reached at bpsdsp@isat.com.

REFERENCES

Anatoniou, A., ***Digital Filters: Analysis and Design,*** McGraw-Hill, New York, N.Y., 1976.

Oppenheim, A.V., and R.W. Schafer, ***Digital Signal Processing,*** Prentice-Hall, Englewood Cliffs, N.J., 1975.

Williams, C.S., ***Designing Digital Filters,*** Prentice-Hall, Englewood Cliffs, N.J., 1986.

SOFTWARE

Software for this article is available through the Circuit Cellar BBS, the Circuit Cellar Web site, and on Software on Disk for this issue. Please see the end of ConnecTime for downloading and ordering information.

SOURCE

Practical Digital Filter Design Manual
DSP Solutions
13269 Sparren Ave.
San Diego, CA 92129
bpsdsp@isat.com

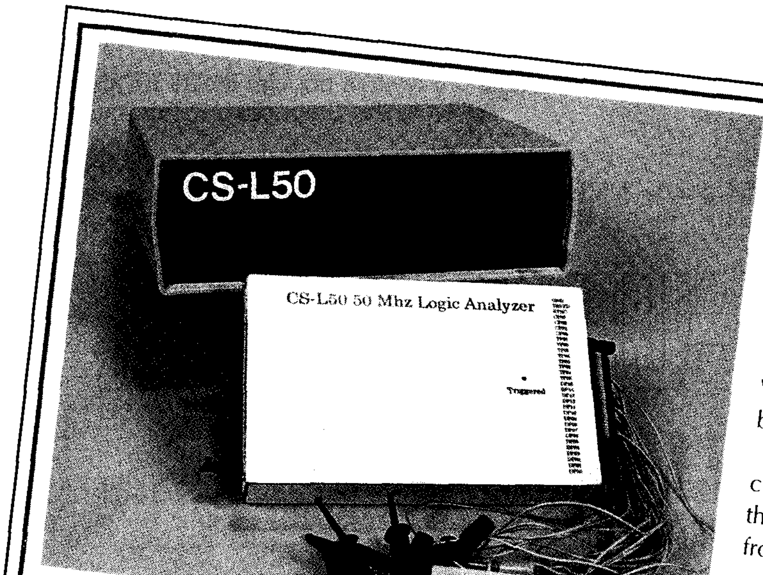
I R S

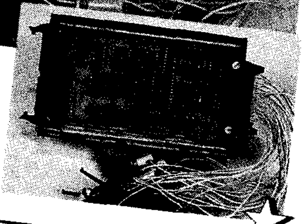
410 Very Useful
411 Moderately Useful
412 Not Useful

8th Annual Circuit Cellar Design Contest Winners

☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆

compiled by Elizabeth Laurençot





☆ 1st Place ☆ Matt Ballance

CS-L50 LOGIC ANALYZER/DIGITAL PATTERN GENERATOR

The CS-L50 is a 16-channel expandable logic analyzer/digital-pattern generator with a 32-Kb record length per channel and data capture rates from 6 to 50 MHz. External clocking is supported with a 4-bit clock/clock qualifier.

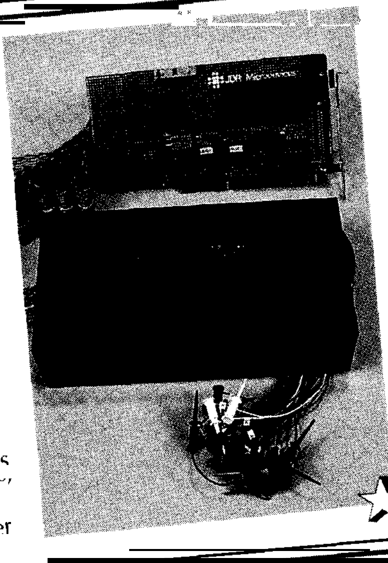
The CS-L50 has eight trigger probes as well as trigger bits for each data bit. Any 8-bit block of trigger bits can produce the main trigger signal and can be selected for input or output. While one block is being used as stimulus signals for the device being tested, another block captures the resulting data.

On the main board, the high-speed block and the 8031 microprocessor capture data from the probes and transfer data to the host via a parallel port. The sampling/trigger pod buffers data from the probes and creates trigger and clock signals.

Additional data capture boards—added via the expansion connector—can operate independently or in tandem with the host or other expansion boards.

You may reach Matt at hardware_hacker@juno.com.

☆ 2nd Place ☆ Robert Morrison



RELOADING STATE ANALYZER

Are you tired of wasting time repeatedly running state analyzers to locate firmware bugs in an embedded processor! If so, Bob designed a small reloading state analyzer that tracks data flow over many possible sequences before initiating tracing.

The analyzer has 131,072 (expandable to 1 million) 16-bit state-memory locations, and it stores up to 65,536 (expandable to 262,144) possible configurations. The user specifies a tree of possible outcomes and appropriate collection of states for all expected instruction sequences. That is, all known possible trigger sequences are stored and the target system is run only once.

The reloading state analyzer—including state and interface, PC interface, match configuration, comparators for 16 bits—is stored in one FPGA. An LED display provides constant feedback of target operation and recording status of the analyzer.

You may reach Bob at rdm@hpbs670.boi.hp.com.

3rd Place

David Gaddis

PROGRAMMABLE DC LOAD

The PDCL tests any DC power source, including batteries, power supplies, and solar cells. It can step or pulse load, hold constant current or power with AC input changes, force constant load vol... ...pply performance under various conditions, and determine current limits and foldback points.

PDCL has a Motorola MC68HC705C8 microcontroller, a Maxim MAX531 voltage output DAC, a Linear Technology LTC1293 12-bit ADC, and an Analog Devices AD633 analog multiplier in an aluminum enclosure. Two PCBs hold all digital and most analog electronic components. A 2 × 16 LCD and full numeric keypad aid the user.

PC control is currently possible, but only as ASCII in and out. David plans to modify the device to operate with a dedicated PC interface program that uses control codes and data packets for faster information exchange.

You may reach David at gaddis@ix.netcom.com.

Honorable Mentions

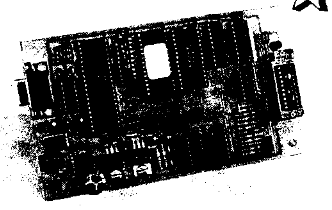
William Rudolph

PRG-2051 DEVICE PROGRAMMER

The PRG-2051 is a low-cost device programmer for Atmel's 89C2051 and 89C1051 flash microcontrollers. The single-board design includes a power supply, and the device is operating-system independent. The user interface shows all device functions in a single help screen.

The PRG-2051 connects to a computer via DE-9 female connector. After setting up a terminal emulator and uploading an Intel hex file, you're ready to program the device. An onboard ZIF socket supports DIP devices.

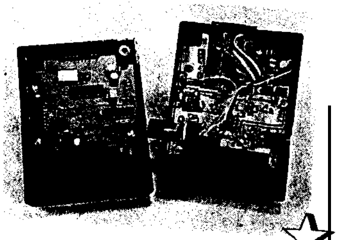
You may reach William at arudoph@edge.net.



John Runciman

POCKET-WATCH TIMER

This timer measures the time-keeping error of pocket watches in plus or minus seconds per day. A clipped-on microphone picks up tick vibrations, which are amplified by a TLC272—a dual low-power, low-noise, CMOS op-amp—and then sent to an LP311 comparator. The comparator output feeds into a 74HC4538, a low-power dual monostable vibrator. John chose a Microchip PICstart 16B programmer and MPLAB to program and debug a PIC16C84 microcontroller for this small, low-power project.



After a 4-s test period, the error displays on the top line of the LCD, while the frequency mode is shown on the bottom line.

You may reach John at john.runciman@worldnet.att.net.

For More Information

Congratulations to all the winners. From analyzers and testers to programmers, the entries encompassed a wide range of projects. No two were alike-and that made judging them all the more difficult.

We encourage all Design Contest winners and entrants to write complete articles about their projects. Design Contest articles are highlighted with the finish-line logo.

If you'd like more information about a project, you may contact its designer via E-mail. Otherwise, you must patiently wait for the full article to appear in an upcoming issue. We will not give out the phone numbers or addresses of the project designers.

If you don't have E-mail access, we'll forward your letter to the designer. Just send it care of:

Design Contest Winner
Circuit Cellar INK
4 Park St.
Vernon, CT 06066

FEATURE ARTICLE

Kenneth Baker

Self-Tuning PD Algorithm for the 68HC11

The day-to-day changes in power and mechanical loads gets controllers out of tune. Ken shows us the math and algorithms behind a self-tuning controller that knows how to maintain efficiency.

Every control engineer is familiar with the problem of tuning the parameters of a digital controller to suit the characteristics of a specific plant. Parameters perfect in one system may be wildly unstable or ineffective in another.

The engineer adjusts the control parameters by trial and error-guided by instinct and experience, sometimes using heuristic rules-until the plant exhibits the desired behavior. The controller is then sent to the plant with the assumption that it will remain reasonably constant.

But, what if that assumption turns out to be false? Many systems have characteristics that change with power supply, mechanical load, or wear on the parts. In these cases, the quality of the control may degrade because the controller is no longer properly tuned.

We need a self-tuning controller that automatically deduces a plant's characteristics and adjusts control parameters to maintain the desired behavior.

The self-tuning control algorithm I present starts with a discrete-time proportional-derivative (PD) control algorithm. Piggybacked on that is a recursive least-squares (RLS) algorithm borrowed from adaptive DSP theory, which estimates the plant's characteristics from its I/O datastreams.

The PD and RLS algorithms are well known and understood. What's not obvious is how to feed the results of the RLS back into the PD.

SYSTEM ANALYSIS

Figure 1 shows a typical system. Signal d(k) is the desired plant output, and y(k) is the actual output. The error signal x(k) is the difference between these outputs. The error signal is usually represented as e(k), but I'm reserving that symbol for something else. The controller output is u(k).

For all signals, I prefer to normalize full scale to 1. That way, the control algorithm can run independent of the application. Other software in the controller can include scaling factors as necessary.

In discrete time, the plant output is described by:

$$y(k+1) = -ay(k) + bu(k) \qquad (1)$$

where a and b are the plant parameters. The controller function is described by:

$$\begin{aligned} &x(k) = d(k) - y(k) \qquad (2) \\ &u(k+1) = u(k) + K_p x(k) + K_d[x(k) - x(k-1)] \end{aligned}$$

where K_p and K_d are the respective proportional and derivative control parameters.

Assuming the controller can maintain this plant at all, the output y(k) eventually reaches a steady state in response to a step input in d(k), at which time x(k) = 0 within some band of tolerance. But, "eventually" doesn't necessarily constitute good, predictable control.

I'd like to optimize the plant's transient response to a step input according to a defined measure by specifying the desired response time and overshoot.

In general, analytical solutions to discrete-time equations aren't possible, so to describe the transient response,

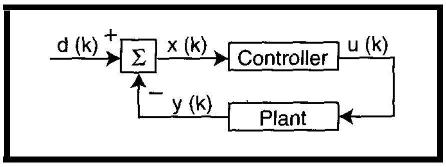


Figure 1—In a typical feedback control system, the target and actual plant outputs are d(k) and y(k), respectively.

Listing I--This *listing shows the combined RLS-PD control algorithm in C.*

```
#include <stdio.h>
#include <stdlib.h>

/* Run-time Signals */
float y, yzl:              /* plant output y(k), y(k - 1) */
float u, uz1;              /* control  signal  u(k), u(k 1) */
float d;                   /* desired (target) plant output */
float x, xzl;              /* rate error x(k), x(k-1) */

/* Controller */
float Kp, Kd;              /* PD control parameters */
float M, coeff:            /* intermediate value for KD and Kd */

/* RLS */
float i1, i2, 1;           /* intermediate values for vector g */
float q1, q2;              /* elements of vector g */
float p11, p12, p21, p22;/* elements of matrix P */
#define INIT_P 1000        /* estimate plant parameters, and */
float a, b;                /* elements of vector w */
float alpha:               /* estimation error */
int rls_cntr;              /* RLS iteration counter */
#define RLS_CNT_VAL 10
float get_actual_rate(void)  /* called function prototypes */
float get_desired_rate(void);
void set_output(float);
void rls_pd_init(void)

  uz1= yzl = xzl  =  0;    /* clear run-time signals */
  rls_cntr = RLS_CNT_VAL;/* initialize RLS */
  a = b = O ;
  coeff = 64 / 49;
  Kp = 1;                  /* set arbitrary PD parameters */
  Kd = 1;

void rls_pd(void)
{                          /* Eq. 2, Control Output */
  y = get_actual_rate(); /* get run-time data */
  d = get_desired_rate();
  x = d - y;               /* calculate rate error signal */
  u += Kp*x + Kd*(x - xz1); /* calculate control signal */
  if (u > 1.00)            /* bound u to 0 to 100% */
    u = 1.00:
  else if (u < 0)
    u = 0:
  set_output(u);           /* set control output */         .
  if (rls_cntr == RLS_CNT_VAL){  /* if time to reinitialize RLS */
    p11 = p22 = INIT_P;
    p12 = p21 = 0;

  if ((--rls_cntr) == 0){/* if time to calculate Kp and Kd */
    rls_cntr = RLS_CNT_VAL;
    M = (a + 1) / b;
    Kp = (64/49)*M*(a+1);
    Kd = M / 7;

  il = p12*uz1 p11*yz1;/*  Eq. 19, Gain vector g */
  i2 = p22*uz1 - p21*yz1;
  1 = 1 + i2*uz1 - i1*yz1;
  gl = i1 / 1;
  g2 = i2 / 1:
  alpha = y + a*yz1 b*uz1; /* Eq. 22, Estimation Error alpha */
  a += alpha*g1;           /* Eq. 21, Estimated Plant Parameters a */
  b += alpha*g2;           /*   and b */
  p11 -= g1*i1;            /* Eq. 20, Inv Correlation Matrix P */
  p12 -= g1*i2;
  p21 -= g2*i1;
  p22 -= g2*i2;
  xzl = x;                 /* z^(-1), Increment Time */
  yzl = y;
  uzl = u;
```

I'll convert equations (1) and (2) to continuous time using these approximations:

$$f_d(k) = f_c(k\tau) \qquad (31$$

$$\dot{f}_c(t) \approx \frac{1}{\tau}[f_d(t+\tau) - f_d(t)]$$
$$\approx \frac{1}{\tau}[f_d(t) - f_d(t-\tau)] \qquad (4)$$

where f_d is the discrete-time-sampled version of the continuous-time variable f_c, and τ is the sample period.

This approximation is not without risk. If the sample period is too long relative to the system's time constants or rise time, the first-derivative approximation is too inaccurate to be useful.

I developed the algorithm on a system with time constants 10–20 times the sample period. In this case, the magnitude and phase distortion introduced by sampling was negligible.

The next step is to convert the continuous-time versions of equations **(1)** and (2) to a single equation describing the response to a step input. By subtracting y(k) and dividing by τ, equation **(1)** can be rearranged as:

$$\frac{1}{\tau}[y(k+1) - y(k)] =$$
$$-\left(\frac{a+1}{\tau}\right)y(k) + \frac{b}{\tau}u(k) \qquad (5)$$

Using the continuous-time approximations in equations (3) and (4), equation (5) becomes:

$$\dot{y}(t) = -\left(\frac{a+1}{\tau}\right)y(t) + \frac{b}{\tau}u(t) \qquad (6)$$

$$\frac{1}{J} = \frac{b}{\tau}$$
$$\frac{F}{J} = \frac{a+1}{\tau} \qquad (7)$$

then equation (6) simplifies to:

$$J\dot{y}(t) = u(t) - Fy(t) \qquad (8)$$

which is the familiar equation for a rotating system relating the moment of inertia (*J*), the coefficient of friction (*F*), and the driving force (u) to rotational speed (y).

To derive the continuous-time approximation to equation (2), first define the continuous-time proportional control parameter $K_{pc} = K_p/\tau$.

Then, by rearranging and dividing by τ, equation (2) becomes:

$$\frac{1}{\tau}[u(k+1)-u(k)]= K_{pc}\,x(k)+\frac{K_d}{\tau}[x(k)-x(k-1)] \qquad (9)$$

Using equations (3) and (4), equation (9) becomes:

$$\dot{u}(t)=K_{pc}\,x(t)+K_d\,\dot{x}(t) \text{ or}$$

$$\dot{u}(t)=K_{pc}d(t)-K_{pc}y(t)+K_d\dot{d}(t)-K_d\dot{y}(t) \qquad \text{(IO)}$$

By differentiating equation (8) and substituting (IO), we obtain the second-order differential equation for the combined plant-controller system:

$$J\ddot{y}(t)+(K_d+F)\dot{y}(t)+K_{pc}y(t)= K_d\dot{d}(t)+K_{pc}d(t) \qquad (11)$$

The transient response of equation **(11)** to a step input needs to be analyzed to quantify the response time and maximum overshoot. To analyze the transient response, take d(t) to be a unit step function and assume y(0) = 0, y'(O) = 0, and d(0) = 0.

Then, taking the LaPlace transform of equation **(11)** and substituting:

$$D(s)=\frac{1}{s}$$

which is the LaPlace transform for a step function of **1, we get:**

$$Y(s)=\frac{K_d s+K_{pc}}{s(Js^2+(K_d+F)s+K_{pc})} \qquad (12)$$

The interesting characteristics of equation (12) can be gleaned by rewriting it as:

$$Y(s)=\left(\frac{\omega_n^2}{c}\right)\left(\frac{s+c}{s(s^2+2\zeta\omega_n s+\omega_n^2)}\right) \qquad (13)$$

where the natural frequency (ω_n), the zero ($-c$), and the damping coefficient (ζ) are defined as:

$$\omega_n=\sqrt{\frac{K_{pc}}{J}}$$
$$c=\frac{K_{pc}}{K_d} \qquad (14)$$
$$\zeta=\frac{K_d+F}{2\sqrt{K\quad J}}$$

For the controller to be self-tuning, it must know what constitutes being in tune.

The desired behavior of the plant must be defined in terms of a set of mathematical criteria that use data available to the controller.

Three criteria to consider are the type of response (underdamped, critically damped, or overdamped), the damping coefficient, and the maximum overshoot to a step input.

If some overshoot can be tolerated, the quickest convergence of the response to the desired value can be achieved with an underdamped system, meaning the damping coefficient is between 0 and **1.**

A damping coefficient close to 0 yields a shorter rise time but greater overshoot. With a damping coefficient close to **1,** overshoot is small but the rise time is longer.

Unless there are other system requirements that affect the selection of the damping coefficient, 0.5 is a good compromise between reducing overshoot and shortening the response.

For a given damping coefficient in an underdamped system, one more factor affects the overshoot: the ratio between the zero (-c) and the real part $(-\zeta\omega_n)$ of the pair of complex poles [1].

If the ratio:

$$\alpha = \frac{c}{\zeta\omega_n}$$

is close to 1, the overshoot can be as high as 70%. Larger ratios reduce the overshoot, and by the time a = 4, the overshoot is down to about 20%.

The minimum possible overshoot is about 14% as a approaches infinite. I used α = 16 to keep overshoot below 20% allowing for some noise, but any number greater than 4 is just as good.

By setting the damping coefficient (ζ) to 0.5, α = 16 reduces to:

$$\sqrt{K_{pc} J} = 8 K_d$$

From this and the definition of ζ in equation (14), it follows that:

$$K_d = \frac{F}{7}$$
$$K_{pc} = \frac{(64F^2)}{(49J)} \qquad (15)$$
$$K_p = \tau \frac{64F^2}{(49J)}$$

RLS ALGORITHM

Now that we can calculate K_p and K_d from the plant characteristics J and F, what next? We know the sample period τ, and J and F are functions of a and b, in equation (6).

So, if we can calculate a and b, we are done. To do that, I returned to the domain of discrete-time and borrowed the RLS algorithm from DSP theory.

Least-squares is a method of finding the best solution to an overdetermined set of simultaneous equations. The solution to the least-squares equation is computationally expensive, involving n-dimensional matrix inversion and multiplication, where n is the number of simultaneous equations.

Such a calculation may be beyond the capability of a simple microprocessor. But, there's a recursive solution to the equation in which a simpler set of calculations is performed for each equation individually, and the results from one set of calculations are fed into the next.

To get the recursive solution, I need to cast the problem into a standard form known as the deterministic normal equation. First define the data vector a:

$$\mathbf{a}^T(k) = [-y(k-1), u(k-1)] \qquad (16)$$

and the estimated parameter vector w:

$$\mathbf{w}^T = [a, b]$$

The performance measure used to judge the estimated parameters is the difference between the measured output y(k) and the predicted output obtained from the estimated parameter vector in equation (1). This estimation error e(k) is:

$$\begin{aligned} e(k) &= y(k) - (-ay(k-1) + bu(k-1)) \\ &= y(k) - \mathbf{a}^T(k)\mathbf{w} \end{aligned}$$

For a datastream of n samples, we get n instances of equation (16), which can be cast into matrix form by defining the data matrix ***A:***

$$\mathbf{A}(k) = \begin{bmatrix} \mathbf{a}^T(k) \\ \vdots \\ \mathbf{a}^T(k-n+1) \end{bmatrix} \qquad (17)$$

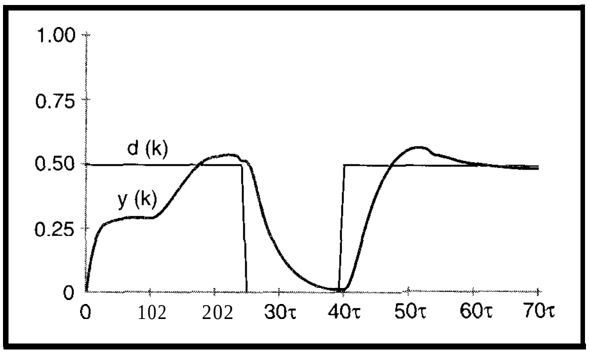


Figure 2-Control is sluggish for the first ten samples *until new control parameters are calculated. At the second sfep input in* **d(k),** *the controller is tuned.*

the error vector e:

$$\mathbf{e}^T(k)=[e(k),\ldots,e(k-n+1)]$$

and the measured data vector y:

$$\mathbf{y}^T(k)=[y(k),\ldots,y(k-n+1)]$$

From these definitions, e equates to:

$$\mathbf{e}(k)=y(k)-a(k)w$$

In the least-squares method, the square of e should be minimized. The squared error function is:

$$E(k)=\mathbf{e}^T(k)\mathbf{e}(k) \\ =\mathbf{y}^T\mathbf{y}-\mathbf{y}^T\mathbf{A}\mathbf{w}-\mathbf{w}^T\mathbf{A}^T\mathbf{y}+\mathbf{w}^T\mathbf{A}^T\mathbf{A}\mathbf{w}$$

The surface defined by this function forms a bowl shape over the *a-b* plane. The *a-b* coordinates under the lowest point in the bowl define the point of minimum error between the measured output and the estimated output calculated from the estimated *a* and b.

At this point, the gradient of the error surface equals zero or:

$$\frac{\partial E}{\partial \mathbf{w}}=-2\mathbf{A}^T\mathbf{y}+2\mathbf{A}^T\mathbf{A}\mathbf{w} \\ =0$$

which reduces to:

$$\mathbf{A}^T\mathbf{y}=\mathbf{A}^T\mathbf{A}\mathbf{w} \qquad (18)$$

This is the deterministic normal equation for least-squares parameter estimation, and the solution w provides the necessary a and b.

Deriving the recursive solution to equation (18) is a rocky trek through linear algebra. But, now that the problem has been made to fit Procrustes-wise into the standard form [2], there's no need to reinvent the wheel. Instead, 1'11 simply assume that a miracle occurs and jump right to the solution.

First, I need to define one new matrix, two vectors, and a scalar. The gain vector g is defined as:

$$\mathbf{g}(k)=\frac{\mathbf{P}(k-1)\mathbf{a}(k)}{1+\mathbf{a}^T(k)\mathbf{P}(k-1)\mathbf{a}(k)} \qquad (19)$$

where the inverse correlation matrix ***P*** is:

$$\mathbf{P}(k)=\mathbf{P}(k-1)-\mathbf{g}(k)\mathbf{a}^T(k)\mathbf{P}(k-1) \qquad (20)$$

The estimated parameter vector w = $[a,b]^T$ is evaluated by:

$$\mathbf{w}(k)=\mathbf{w}(k-1)+\mathbf{g}(k)\alpha(k) \qquad (21)$$

where the estimation error α is:

$$\alpha(k)=y(k)-\mathbf{a}^T(k)\mathbf{w}(k-1) \\ =y(k)-\mathbf{w}^T(k-1)\mathbf{a}(k) \qquad (22)$$

The RLS parameter estimation algorithm proceeds as follows. First, equation (19) calculates the gain vector g, and equation (22) calculates the estimation error (α).

Using g and a, the estimated parameter vector w is recursively updated in equation (21). In the last step, equation (20) recursively updates *P*. The algorithm is initialized at $k=0$ with $\mathbf{w}(0)=0$, and $\mathbf{P}(0)=p\mathbf{I}$, where *p* is an arbitrary number greater than 1.

Once a and b are estimated, the control parameters K_p and K_d are found from equations (7) and (15), which reduce to:

$$K_d=\frac{(a+1)}{7b} \\ K_p=\left(\frac{64}{49}\right)\frac{(a+1)^2}{b} \qquad (23)$$

In equation (23), the control parameters seem independent of the sample rate, and in a sense they are. The sample rate doesn't directly contribute to the control parameters, but a.different sample rate would be made manifest by different plant parameters a and *b.*

PD_RLS CONTROL ALGORITHM

Several issues need to be considered before the RLS algorithm is ready. One is the value of *p* used to initialize ***p***(0).

The only constraint on the value of p-that the correlation matrix $P^{-1}(k)$ has to stay invertible-is met by setting *p* to a number greater than 1 [3]. Otherwise, p's value has little effect on the results, except that a larger number results in faster convergence.

Another consideration is when to recalculate the control parameters K_p and K_d. The estimated plant parameter vector w is inaccurate for a short time after initialization. Control parameters calculated from immature parameter estimates don't result in good control.

In a noisy system with a constant target rate, the system reaches a steady state with the error signal equal to zero plus a noise component. In this case, the RLS algorithm can get confused and calculate plant parameters that reflect the noise more than the true plant characteristics.

You could skip recalculating the estimated plant parameters unless the target rate changes by an amount sig-

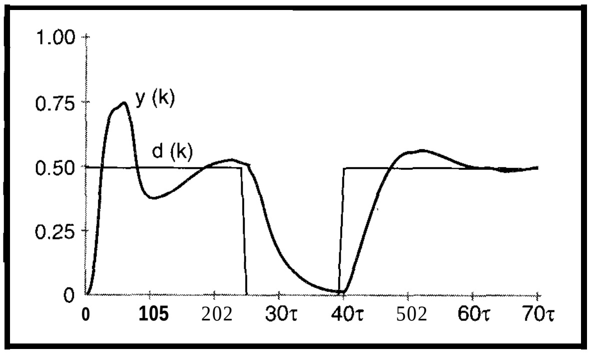


Figure 3-At *first, the output wildly overshoots. Again, the controller is tuned at the second step input.*

nificantly greater than the magnitude of the noise.

I found empirically that I can obtain good results by running ten iterations of the RLS algorithm before using the estimated plant parameters to recalculate the control parameters. Then, I reinitialize the RLS algorithm and recalculate the control parameters after every ten iterations to keep up with changing plant characteristics.

Listing 1 shows the combined PD-RLS control algorithm in C. The intermediate vector $\boldsymbol{i}$ used in calculating $\boldsymbol{g}$ is also used in the calculation of $\boldsymbol{P}$. This works because $\boldsymbol{i} = \boldsymbol{PA}$ in (19), so $\boldsymbol{i}^T = \boldsymbol{A}^T\boldsymbol{P}$, which is what (20) needs.

SYSTEMSUCCESS

Figures 2 and 3 show data obtained from trial runs on the same system. In each case, the target rate is 0.5 from time 0 to time 2.5 s, after which it goes to 0. At 4 s, it returns to 0.5 again.

In Figure 2, the control parameters start at $K_p = 0.001$ and $K_d = 1$. The result is a sluggish response for the first second until the control parameters are recalculated. The response to the input pulse at 4 s reaches the target rate within 1 s, overshoots by about 10%, and then settles down.

In Figure 3, the control parameters start at $K_p = 1$ and $K_d = 0.001$, which results in almost 50% overshoot in the first second. But again, the response to the input pulse at 4 s is well-behaved.

COMPUTERS IN THE FIELD?

This algorithm allows the controller to calculate automatically the parameters needed to achieve the desired response. It produces a transient response with specific characteristics, but other characteristics can be chosen. Different combinations of the damping coefficient and the zero in equation (13) simply change the constants used in equation (20).

I implemented this algorithm on a 68HC11 controller with a 12-MHz clock. I was able to use a sample period of 100 ms with plenty of time available for other system functions.

The controller was used on a fertilizer spreader-literally, "in the field"—in which the fertilizer application rate varied with the speed of the truck through the field to apply a constant amount of fertilizer per unit area.

To complicate the issue, the machinery's response changed with power-supply voltage, load, and usage as bearings and gears jammed with fertilizer and dust. Also, the operators wanted to use the same controller on several different systems and to switch from one to another at any time without retuning the controller.

The algorithm proved to be robust. The controller could connect to a new system and be tuned in l-2 s. And, the tuning changed with plant characteristics, keeping response consistent. ■

Ken Baker works as a senior design engineer at Guidant/CPI, a medical-device company. You may reach Ken at qc03660@stp03.guidant.com.

SOFTWARE

The algorithm in 68HC 11 assembly code, available on the Circuit Cellar BBS, makes use of floating point functions available on the Internet at http://freeware.aus.sps.mot.com:80/freeweb/amcu_ndx.html#mcu11.HC11FP11.Asm is freeware maintained by Motorola.

REFERENCES

[1] K. Ogata, ***Modern Control Engineering,*** Prentice Hall, Engle-

[2] Haykin, ***Adaptive Filte***
Prentice Hall, Englewood
Cliffs, NJ, 381, 1986.

[3] ***Digital Control Sys-***
Springer-Verlag, Berlin,
367, 1989.

SOURCES

68HCll
Motorola
MCU Information Line
P.O. Box 13026
Austin, TX 7871 l-3026
(512) 328-2268
Fax: (512) 891-4465

I R S

413 Very Useful
414 Moderately Useful
415 Not Useful

FEATURE ARTICLE

David Rector

Getting Started With Xilinx EPLDs

Part 3: Hands-On Project-Implementation

In Part 1, we learned the benefits of designing with EPLDs. In Part 2, a timer/counter circuit was connected to an EPLD. Here, David demonstrates how the EPLD removes the guesswork from your logic designs.

After you read this series, I'll bet you never design with discrete gate components again. Erasable Programmable Logic Devices (EPLDs} contain every imaginable logic configuration bundled into a single chip.

In Parts 1 and 2, Conrad and I described the implementation of a design into a Xilinx EPLD chip. If you're on a tight budget, skip the schematics in Part 2, download the equation files, and go on to implementation with inexpensive implementation software.

In Part 2, I described the concept and design of a timer/counter circuit with microsecond accuracy that produced an analog waveform representing the counter's current status. Many applications use an analog waveform (i.e., AIL code) to synchronize recorded data, especially when many analog waveforms are written onto a strip-chart recorder.

In this final installment, I'll tell you how to connect the chip into a circuit that plugs into an IBM-compatible computer. Then, you'll be able to program and use the timer/counter/waveform circuit. But first, you must select the appropriate chip for the design,

CHOOSING SILICON

Speed isn't a major concern-I need l-MHz clock rates-but future expansion to finer resolution waveforms makes the EPLD's high speed essential over FPGAs (up to 17 MHz for this chip and up to 83 MHz with a faster EPLD and less functionality in the counter/address generator).

First, estimate the size of the chip. Xilinx makes chips with several gate densities and in various packages—from the XC73 18 (with 18 macrocells) to the XC73144 (with 144 macrocells).

Count how many flip-flops you need in your circuit. You need at least one macrocell for each flip-flop or output equation.

The AIL-chip design has 82 flip-flops, so I chose the '73108 series. If

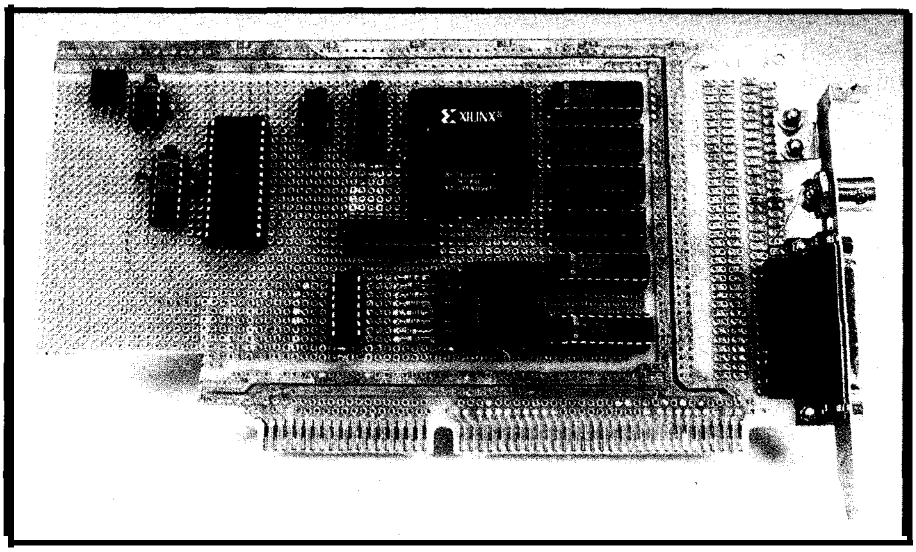


Photo 1-A complete *waveform generator/timer/counter card fits on a half-size PC card with plenty of room to experiment.*

you have complicated logic to implement, as I did, some equations are split into more than one macrocell.

Recall that 18 of the macrocells of the '73 108 are in Fast Function Blocks (FFBs). They operate faster but implement less logic (see Part 2, *INK 75).*

The XC73 18 and XC7336 are made exclusively from FFBs. I chose the XC7336 for the ISA interface because of the high speed of the FFBs and because of the number of I/O pins.

Finally, count the number of I/O, 0, and I pins you need to get signals on and off the chip. The AIL-chip design requires few pins, so the PLCC 84-pin package was enough. If you need more external pins, go to a higher-pin package (e.g., the QFP 160).

SELECTING SPEED GRADE

To make a microsecond counter, the chip must operate at 1 MHz. Predictable timing between inputs and outputs is another advantage of EPLDs over FPGAs.

The Xilinx databook lists timing specifications for each of the five different speed grades for their chips. There is a 5-, 7-, 10-, 12-, and 15-ns guaranteed delay from the fast clock input to output valid.

Keep in mind that, if your output equations get complicated, multiple macrocells cascade with increased delay. Some paths are faster if they pass exclusively through FFBs.

The AIL-chip design requires at most 1000 ns from fast clock input transition to outputs valid (count time), so the 15-ns XC73108 is plenty fast for my project. The exact timing specifications are in the timing report.

The 15-ns chip operates at 17 MHz, and the 7-ns chip at 34 MHz. If you simplify the circuit by cutting out the 999,999-µs reset and decade seconds counters, you could scream at 83.3 MHz.

For the ISA interface chip, the bus specifications list 50-ns maximum delay for zero-wait-state communication with a 16-MHz bus clock. If you ever wanted to hook up to a PCI interface-which is easy with a PCI add-on chip-you can't tolerate more than 20-ns delay for 33-MHz operation.

Most of my PC-interface designs target the PCI bus, so I required the speed of the -7 grade XC7336. If you stick exclusively to ISA, you can spend less with the XC7336-15.

Determining your design's speed can be tricky if there are multiple macrocell levels. And, various signal paths operate with longer or shorter delays, depending on how the fitting software implements your circuit.

Part Name	# of Outputs	# of Input Lines Used	Signal Inputs	# of Shared Pt	O/IO Req.	O/IO Avail.	Size Factor
AIL chip:							
FB1	7	24	24	0	6/0	8/0	9
FB2	9	21	21	0	8/0	8/0	9
FB3	9	21	26	3	0/0	0/0	9
FB4	9	19	19	0	0/0	0/0	9
FB5	9	2i	29	0	2/0	2/0	9
FB6	9	21	21	11	9/0	019	9
FB7	9	20	26	1	9/0	0/9	9
FB8	9	19	19	0	0/0	0/9	9
FB9	9	19	19	0	0/0	0/9	9
FB10	7	21	28	1	3/0	2/1	9
FB1 1	9	20	28	2	0/0	0/0	9
FB12	9	21	21	2	0/0	0/0	9
AILISA:							
FB1		2	2	0	0/0	019	1
FB2	9	19	19	0	8/0	1/8	9
FB3	9	18	18	0	8/0	3/5	9
FB4	0	0	0	0	0/0	0/9	0

Table l--The *fitting report* describes how well your design fits into the *EPLD.* Unused *macro*cells *are the key to how much room you have for expansion. Nearly all of the 108 outputs—each representing a single macrocell-are used in the All-chip implementation. The AILISA chip is only ha/f utilized, so you can improve performance by making a synchronous interface.*

Look closely at the Xilinx databook For the XC73108-7, if you use the fast clock to drive a flip-flop, there's a maximum of 7-ns delay. But, the same flip-flop driven with a regular input as the clock line gives you a 13.5-ns delay. For a pure logic equation with no flip-flops, there's a 16.5-ns delay from input to output.

PROGRAMMING EPLDS

After your design is drawn into the schematic-capture software or you have written the equation files, you can program the logic in the EPLD.

Once you have a general idea about which chip to target, run Xilinx's design manager to see how everything turns out. The reports provided by Xilinx's implementation program are very good at describing the resources needed and the resulting timing.

If cash is limited, bypass the schematic-entry phase and download the equation files (AILCHIP.XFF and AILISA.XFF). Use Xilinx's DS-550 design-manager software to generate the programming files (AILCHIP.PRG and AILISA.PRG).

EPLDs are programmed in three steps. After the schematic is entered, the design is translated into an equation file. The Xilinx stand-alone base system comes with a Design Manager program that performs this operation and is similar to the DS-550 described in Part 1 *(INK 74).*

The Windows-based software is amazingly easy to use. After specifying the AIL-chip design in New Project, click Translate, and choose the XC7000 family.

The translation routine comes up with a device-specification window. You must select XC73108-15PC84 in the device specification for the AIL-chip design if you haven't entered the device in the schematic. For the AILISA interface chip, use XC7336-7PC44.

Next, implement the equation file for the chip. The implementation routine fits the equations on the chip and produces various performance reports and timing files.

Try implementing the AIL chip with different speed grades to see what you get in the timing reports. Use all the default options.

If you're packing a lot of logic into a chip, turn the timing optimization off or your design won't fit. Of course, you can try having the timing optimization on. Your design runs much faster, but more resources are required.

If you've wired the chip and need to freeze the pins, select Guide Design and choose the last version that uses your pin configuration. Usually, you can freeze the pins without much trouble.

AIL Chip

Pkg Pin	Pin Type	Pin Use	Pin Name
1	MR		
2	I	tie	(unused)
3	I	tie	(unused)
4	I	tie	(unused)
5	I	tie	(unused)
6	I	tie	(unused)
7	I	tie	(unused)
8	VSS		
9	CLK	I	CLK
10	CLK	0	MA13
11	I/O	0	MA5
		O	MA15
12	COK	O	DHB6
13	O	O	DLBO
15	0	O	DLB1
16	vss		
	O	O	DLB2
18	O	O	DLB3
19	O	O	DLB7
20	O	NC	(unused)
21	0	NC	(unused)
22	vcc		
		(O)	(SDH0_INV)
23	I/O	(O)	(SDH1_INV)
		(O)	(SDH2_INV)
26	I/O	(O)	(SDL1_INV)
27	vss		
		(O)	(SDL10_INV)
29	I/O	(O)	(SDL11_INV)
		(O)	(SDL12_INV)
30	I/O	(O)	(SDL13_INV)
		(O)	(SDL14_INV)
33	I/O	(O)	(SDL15_INV)
		(O)	(SDH3_INV)
35	I/O	(O)	(SDL0_INV)
36	I/O	(O)	(SDL2_INV)
37	I/O	(O)	(SDL3_INV)
38	VSS		
39	I/O	(O)	(SDL4_INV)
40	I/O	(0)	(SDL5_INV)
	I/O	(O)	(SDL6_INV)
42	VSS		
43	I/O	(O)	(SDL7_INV)
44	I/O	O	DHB0

AIL Chip

Pkg Pin	Pin Type	Pin Use	Pin Name
45	I/O	O	DHB1
46	I/O	O	MA10
47	I/O		MA1 1
48	I/O	0	MA12
49	vss		
50	I/O	0	MA18
51	I/O	0	MA2
52	I/O	0	MA17
53	I/O	O	MA19
54	I/O	O	MA8
55	I/O	O	MA20
56	I/O	O	MAO
57	I/O	O	MA1
58	I/O		MA3
59	I/O	0	MA16
60	VSS	O	
61	I/O	O	MA4
62	I/O		MA6
63	I/O	0	MA7
64	VCC		
65	0	0	DLB6
66	O	O	DLB5
67	O	O	DLB4
68	0	0	DHB7
69	O	0	DHBS
70	O	O	DHB4
71	0	0	DHB3
72	0	0	DHB2
73	VCC		
74	CEN	0	MA14
75	CEN	0	MA9
76	FOE	I	EN_LB
77	FOE	I	EN_HB
78	vcc		
79	I	I	A2
80	I	I	MAH_EN
81	I	I	MAL_EN
82	I	I	LREQ
83	I	I	C_EN
84	I	I	AI

AILISA

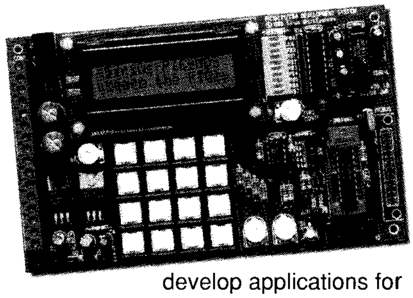
Pkg Pin	Pin Type	Pin Use	Pin Name
1	MR		
2	I/O	0	*IOW1
3	I/.O	0	*IOW0
4	I/O	0	*IOR3
		O	*IOR1
6	CLK	O	*IOR0
		O	*IOR
8	I/O	O	*IOW
9	I/O	I	AD3
10	VSS		
11	I/O	I	'BALE
12	I/O	I	*AEN
13	I/O	I	AD4
14	I/O	I	AD5
15	I/O	I	AD6
16	I/O	I	AD7
17	I/O	tie	(unused)
18	I/O	tie	(unused)
19	I/O	I	AS8
20	I/O	I	AS8
21	vcc		
22	I/O	I	AS6
23	VSS		
24	I/O	I	AS5
25	I/O	I	AS4
26	I/O	I	AD9
27	I/O	I	AD8
28	I	I	AD1
29	I/O	I	ADO
30	I/O	0	EN_LO
31	vss		
32	VCC		
		O	EN_HI
33	I/O	O	*DQ_EN
35	I/O	0	DIR
		O	*AM_WR
36	I/O	O	*AM_EN
		O	AIL_A2
38	FOE	O	AIL_A1
40	FOE	0	*IOR2
41	vcc		
42	I	I	AD2
43	I/O	O	*IOW3
44	I/O	O	*IOW2

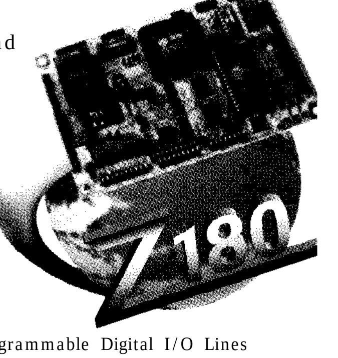
Table 2—*The pinout report lists each pin on the* **device, and how it is used. Some abbreviations are** *I-L* **=** *input uses latch, I-R =* **input uses register,** *I/O-L* **=** *I/O* **uses latch,** *I/O-R* **=** *I/O* **uses register, NC = not connected/not available, tie = unused** V_{CC}**st be***GND,***to** *attached* **to us** *macrocell* **gefthiswhenyouusethedrive** *option* *R(MR)* *1.*

If you've made some major design changes, conserve some pins by modifyingAILCHIP.GYD.Thistextfileputs specific equations into specific macrocells and directs I/O to specific pins.

Find the .GYD file you want as the base and delete the equation lines and any pin specifications that may make it difficult to fit the new design. If everything goes well, you'll get a .PRG file with all the report files. Put the chip in the ZIF socket of the Deus Ex Machina programmer and run XPGM.

The software provided with the XPGM programmer needs to know what kind of chip you're programming and the image file name. For the AIL-chip design, use this command to select the device, the package, and the input file to program (see Part 1):





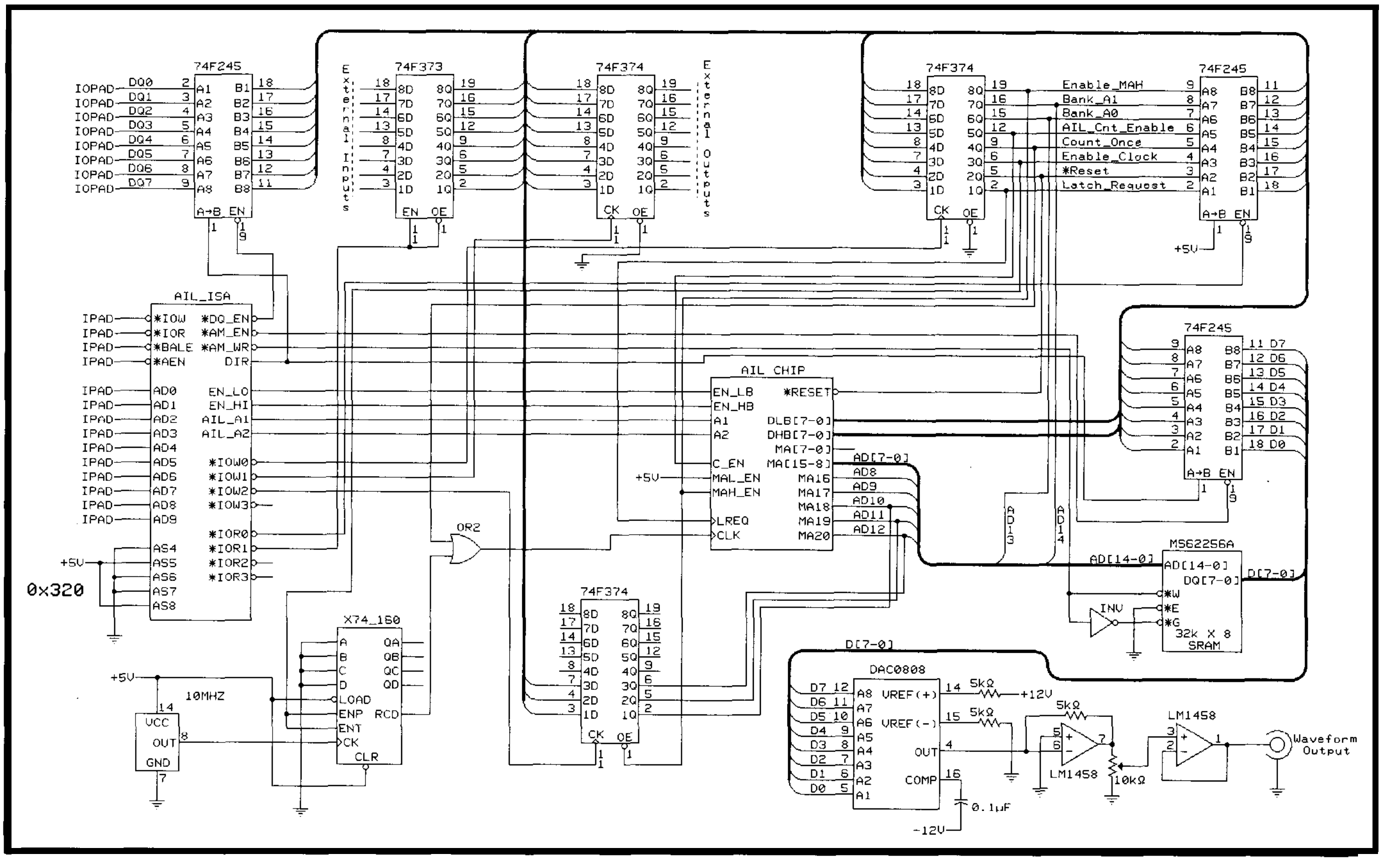


Figure 1—*The AIL-chip EPLD* can be connected to an ISA-bus *interface* of an *IBM-compatible* computer. The *AILISA* chip does *all* of the address decoding, and three *74F374s* register the *I/O port* data for three of the addresses. *If* you get a *1-MHz* clock chip, you can cut the *74F160* decade counter.

```
XPGM -d XC73108 -p PLCC84 -i
   ailchip.prg -P
```

For the AILISA chip, use:

```
XPGM -d XC7336 -pPLCC44 -i
   ailisa.prg -P
```

If you need to erase your EPLDs, be sure to erase the chip completely. Incomplete erasure is not always indicated by a blank check (see Part 1). It is, however, indicated by erratic outputs [usually held high) or high running temperature (so hot you can't hold your finger on it).

FITTING REPORT

The fitting report describes how the EPLD components are used. Each block has nine outputs. You can see that nearly all of the outputs are used.

Your design doesn't fit if you get Overflow function blocks in the fitting report. If you can't fit your design into a particular chip, but you know you're close, try these tricks.

Underutilized FFBs are solved by paying strict attention to the rules. I used only fast clock lines to drive flip-flops. If any outputs are tristated, the enable line must be driven with the special BUFFOE line.

If you have one or more regular function blocks with fewer than nine outputs used and few outputs on the Overflow function blocks (i.e., the total number of outputs doesn't exceed the available macrocells), you have too many inputs on some equations.

You can split complicated equations in your circuit by placing a BUF symbol between logic levels. This technique increases delay time but lets you pack more logic into the chip.

Table 1 summarizes the fitting report for the AIL chip. Much more information is provided, including how macrocells were assigned to each equation. Any tricks the program played while you weren't optimizing performance are also listed here.

PINOUT REPORT

The pinout report is straightforward (see Table 2). If you were careful about labeling all of your nets (i.e., wires on the schematic), these names appear on the pinout report for each output and input.

It also lists the power and ground lines and instructs you about the unused lines. Some can be ignored depending on how you set the **Drive Unused I/0** option, and some have to be tied high or low.

The Master Reset line is always on pin 1. This pin is useful if you're short on space-but be careful. If the flip-flops are reset by the same signal, you can tie all of them low and use the Master Reset line to reset the chip instead of using macrocell resources.

This technique can significantly reduce the amount of logic needed, but depending on how the flip-flop is implemented, it does different things. If the flip-flop is part of an output latch, it resets high. If it's inside the macrocell, it resets low.

Also, the Master Reset pin can take a few milliseconds to complete the chip initialization. Be prepared to wait.

Lastly, on chips like the XC7336 (not the XC73108], the Master Reset pin can be used as a regular input signal pin. This must be specified either in the schematic or the implementation options.

TIMING REPORT

As far as your design's performance is concerned, the timing report is the most important and most useful. It tells you if your design will fly in the real world.

If your clock is going at 20 MHz and the timing report says 1 7-MHz maximum external clock rate, you better use a faster chip. Otherwise, the behavior is unpredictable.

The timing report summarizes the signal delays between input and output pins, as well as delays from clock edges to output valid.

If your design isn't complicated or if you used the optimize timing option during the implementation, the timing report should merely be a copy of the databook timing specifications for your chip. Otherwise, this report shows how the split equations are going to affect the timing delays.

For the AIL-chip design, I suffered a significant loss in speed due to my split equations, since my maximum external clock rate is 17 MHz. With no split equations, I'd expect 45 MHz.

The I/O delays also help assess how the design performs in my circuit. For example, a change at pad Al to select the output byte takes, at most, 36 ns until the output data bus is stable. This is adequate for an ISA zero wait-state interface running at 16 MHz, but not fast enough for a 33-MHz PCI bus transfer.

BOARD CONSTRUCTION

Once your chips are programmed, you can construct the PC board. I recommend programming the chips before wiring the sockets because if you have trouble implementing the design, you may want to change the chip's pin configuration.

Figure 1 illustrates the schematic of the completed circuit. The AILISA chip {XC7336-7PC44} handles all of the glue logic between the host computer and the rest of the board.

A 74F245 acts as the data bus buffer, another 74F245 buffers data to program and reads the SRAM chip, and a 74F373 latches external inputs. Three 74F374s register control outputs, and a third 74F245 reads back one of the control registers.

One of the three output registers controls AIL-chip operation, such as latch request, reset, single stepping, waveform-bank selection, and SRAM programming. The second output register drives the SRAM upper bits during programming, and a third output register drives external outputs.

I inadvertently discovered that the EPLDs are terrible at implementing the registered output signals of the 74F374s. Unfortunately, Xilinx doesn't provide enough ground pins, so occasional spikes may occur on output lines not registered by the fast clock signals.

So, if any of your static outputs (e.g., LREQ) drive a clock or enable line which isn't in sync with the fast clock line of the signal source EPLD, you may get an occasional erroneous trigger on that line from ground spikes within the signal source EPLD chip.

As a rule, I always register my static output lines with something like a 74F374 and leave the signals governed by the master clock signal to the EPLD. As long as the output signals are synchronous with the clock signal, I haven't had problems with the EPLD.

As you can see in Photo 1, the entire circuit fits onto a half-size PC board. Of 15 chips, 2 are used for external I/O and are not needed for the AIL circuit. The EPLD implementation also requires fewer wires than discrete components, which cuts down on construction time and error.

WAVEFORM GENERATION

The address lines MAO-MA20 from the AIL chip drive the address inputs of an SRAM chip. In this example, lines MAO-MA7 are ignored, which is too fine of a resolution for right now. So, MA8–MA20 connect to address lines AO-A12 of a 32-KB SRAM chip.

For my design, I use the upper two bits of the SRAM address space (Al3 and A14) as waveform-bank storage. You can select up to four different waveforms by setting these bits of the control I/O. The SRAM can be programmed by cycling through timer/counter addresses with the single-step control and writing to the SRAM.

The data lines of the SRAM chip drive an 8-bit DAC0808 and amplifier. This example only updates the DAC every 0.25 ms, so a slow 85-ns SRAM and cheap DAC are adequate.

For higher performance waveform generation-one that updates each microsecond, or up to 17 MHz for the 15-ns EPLD-use a good 25-ns SRAM chip with a latchable DAC (e.g., DAC-0830) and drive the DAC latch with the timer clock. For easier SRAM programming, I added tristate controls of the MA address lines on the AIL chip.

The waveform file used to program the SRAM has four 5 12-byte-long sequences. Sequence 0 is a negative-going deflection. Sequence 1 is a positive deflection with a small amplitude hump, and sequence 2 is a positive deflection with a tall hump. Sequence 3 is a baseline, flat level. The AIL code is made up of 80 of these 512-byte sequences every 10 s.

The SRAM address-generator circuitry determines the sequence display order. Every 10 s, a negative deflection is followed by a positive deflection, which is short or tall depending on the state of S19.

Another negative deflection is followed by a positive deflection set by S18. This process continues until all the second bit states are displayed (excluding SO-S3).

Between each BCD number, a baseline level separates seconds digits, and between each 10 s are more baseline-level sequences. More specifically, whenever SO or S3 is high, a baseline-level sequence results.

Whenever U17 is low, we get a negative deflection. Whenever U17 is high, we get a positive deflection set to the state of the seconds-counter bit selected by U18, U19, S1, and S2.

Look at the SRAM address-generator circuit to see how this is done. I wrote a C program and a sample waveform file for you to test your circuit.

SOFTWARE TESTS BOARD

This program resets your AIL chip, loads the waveform file into the SRAM chip, continuously reads the AIL chip, and displays the counter's status. I list the I/O ports you need and describe each one in the program.

The program requires you to enter the I/O port address of your board as a parameter when starting the program. The file AI LWAVE. DAT must be in the same directory for the waveform to be loaded.

To load the waveform into the SRAM, the program cycles through the microseconds counts in single-step mode to generate the lower 10 address bits. The upper three bits are set by clearing bit 7 of the AIL control I/O port which disables the upper three bits from the AIL chip and drives them independently.

Since the SRAM ignores the lower 8 bits of the microseconds, it writes a value to the SRAM only after every 256 steps. Based on the state of U17 and the upper 3 bits of the SRAM address IO port, a value from the waveform sequence is written to the SRAM.

Once the waveform is programmed, the program's second part continuously latches, reads the AIL count state, and calculates the time between

reads. It does a line feed if something is wrong with the time difference.

The AI L-RESET routine clears the Master Reset pin and waits for the LREQ latch to go low (i.e., not ready on the UH14 line). This change indicates that the chip is finished resetting and can take as long as a millisecond.

NO MORE GUESSWORK

Now, **you're** ready for your own design. Once you get through your first EPLD implementation, logic design is a breeze. No more guessing about how discrete components will interact.

If you perfect the art of synchronous design, you'll eliminate race conditions. You can sit back in your most comfortable chair and simulate the design until you're sure it works. Then, burn the part and wire it up.

I think discrete logic gates will soon be obsolete, so I hope you're now feeling comfortable with the next generation of logic design tools. ❑

David Rector is a post-doctoral fellow in the neuroscience program at UCLA. He is continuing research on imaging light-scattering changes in brain tissue. You may reach David at *dave@aunix.loni.ucla.edu.*

SOFTWARE

An AIL-chip test software sample is available via the Circuit Cellar BBS, the Circuit Cellar Web site, and Software on Disk for this issue. Please see the end of ConnecTime for downloading and ordering information.

SOURCES

SRAMs
Nu Horizons
31225 La Baya Dr., Ste. 104
Westlake Village, CA 91362
(818) 889-9911
Fax: (818) 889-1771

Marvac Dow Electronics
2001 Harbor Blvd.
Costa Mesa, CA 92627
(714) 650-2001
Fax: (714) 642-0148

DS-550, XACT 6.0-DSVL DSVL, FPGA, EPLD Data Book
Xilinx Corp.
2100 Logic Dr.
San Jose, CA 95 124
(408) 559-7778
Fax: (408) 559-7114
http://www.xilinx.com/

Deus Ex Machina programmer, Xilinx parts, and software
Digi-Key Corp.
701 Brooks Ave. S
Thief River Falls, MN 56701-0677
(218) 681-6674
Fax: (218) 681-3380

EPLD programmer
Deus Ex Machina
1377 Spencer Rd. W
Saint Paul, MN 55108
(612) 645-8088
Fax: (612) 645-0184

IRS

416 Very Useful
417 Moderately Useful
418 Not Useful

The Thrifty Engineer

FEATURE ARTICLE

Daniel Nygren

Although Dan's not a lover of the embedded PC, he finds an ideal solution for one here. If you too have been unable to junk that old AT, perhaps you need to join Dan in a lesson on effective recycling.

being thrifty comes naturally to me. I was born tight as a ‘OS-mm pitch quad flatpack, and this carries over to my work as an engineer. This trait-and a certain pack-ratish quality-comes in handy when I'm cobbling together a circuit to meet a deadline.

The project I'm about to share with you is atypical for me because it involves an IBM PC clone. Most hardware types like myself sneer at PCs for embedded systems because 80x86 assembly language is painful, the ISA bus stinks, and MS-DOS is neither real time nor multitasking.

These problems-and the fact that the great unwashed think a PC can do anything, and if it can't, just get a faster one-tend to make engineers overlook PCs, if only at first.

REVIEW

Here's the project in a nutshell. I needed to take in a continuous stream of asynchronous data at 4800 bps with RS-422 levels, interpret it, and display it on a number of standard composite video monitors.

Well, I helped design a Motorola 68302 async/sync RS-232/RS-422 protocol conversion board, but it had no video output. I considered taking the data in on one RS-422 port on the 68302, formatting it, and sending what was destined to be composite video out RS-232 to an old Micromint Termite terminal or other RS-232-to-video board. But, my cheaper instincts beckoned me elsewhere.

We had an abandoned 19" rack-mount IBM PC/AT clone lying around our lab. It dawned on me that the IBM PC/XT CGA cards had an RCA jack that was an composite video output. I set up the little-AT-that-could, ripped out its VGA card, and slapped the CGA card into an 8-bit slot.

Warning-while genuine IBM PC/ATs have two 8-bit expansion slots, not all AT clones have them. Most CGA cards are of the full-length variety that won't fit into a 16-bit slot because the lower part of their circuit board hangs down too low.

I wrote a program in Borland's Turbo C++ V. 3.0 for DOS after a cursory inspection of its built-in communication routines. A tight loop of C code continuously polled the communications port and printed the incoming characters on the screen.

Well, the little AT couldn't. It dropped incoming characters regularly. Of course, my final program didn't have to print to the screen continuously, but I considered it a good benchmark to see what the system could do.

REPAST

Over lunch at the local squat 'n' gobble, I whined to a couple engineers about my predicament and how if IBM had wanted PC design done right, they should have asked me. After the one-sided diatribe, one of my buddies said he knew someone who used a PC to do communications programs with some canned C/C++ callable routines.

So, I hunted this guy down. He told me he'd used Blaise Computing's C Async Manager with success. The subroutines he purchased implement an interrupt-driven communication port buffer that stores up characters until a main routine is ready for them. Since he was done with the program, a quick license check followed by erasing the files off his hard disk meant I was back on track.

If you're still too cheap to spring for a nice package like Blaise Computing puts out, don't despair. Similar subroutines are available on the Internet.

All I needed was a way to get asynchronous RS-422 data into my PC. I contemplated using a couple of Maxim MAX233 dual RS-232 transmitters and receivers chips and some AMD AM-26LS3 1 and AM26LS32 quad RS-422 line drivers and receivers on a spare

Listing 1—An array of character *pointers is used to store the string indicating the location of the handset on each circuit. When a handsef comes off the hook, the string displays on the appropriate conference.*

```
char * location[MAX_CIRC]={

// The string to be displayed when a circuit joins a conference
// is held in this array. The length of the message in quotation
// marks is fixed at 19 chars. Start with a space and put in 4
// chars and a space for the number. This leaves 13 for the name.

// (HANDSET NUMBER) NAME CIRCUIT #
//  " 19 chars always! "   //comment

    "(18) EW OPS 1      ", //circuit 0
    "(19) EW PED        ", //circuit 1
    "(64) CIRCUIT 2     ", //circuit 2
    "(XX) CIRCUIT 3     ". //circuit 3
    "(73) DATA 3        ", //circuit 4
    "(XX) DATA 3        ", //circuit 5
    "(15) RADAR 15      ", //circuit 6
    "(14) RADAR 14      ", //circuit 7
    "(63) CTRL 3        ", //circuit 8
    "(XX) CTRL 3        ", //circuit 9
    "(74) RM 13         ", //circuit 10
    "(XX) RM 13
    "(12) RADAR 12      "
    "(16) RADAR 16
    "(62) CTRL 2
    "(XX) CTRL 2
    "(09) PDPS
    "(08) EW ACQ
    "(70) SAFTEY
    "(XX) SAFTEY
    "(20) MIR 2         ", //circuit 20



    "(67) GRUM          ", //circuit 140
    "(68) PES 2         "
    " (69) T2-31        ",
    "(75) MATS          "  l/circuit 143
```

XT bus wire-wrap board to do a quick and dirty level conversion to RS-232. That way, I could use a standard communication port board.

But, it seemed kludgy to have the RS-422 signals come into the PC, be converted on my wire-wrap board, and looped back out, only to be taken in by a communication port. There had to be a better way.

I just happened to get a catalog of PC data-acquisition boards from Quatech earlier that week. I paged through until I found a nice RS-232/RS-422 asynchronous communication board, the RW-100TS. For a few extra bucks, I could get it with 16550A UARTs instead of the common 16450/8250 chips.

The 16550As have a 16-byte receive FIFO that the Blaise Computing routines can take advantage of. I thought I'd get an extra margin of safety in case my code got more complicated than I thought, so I sprang for the board with the 16550s to make my major expense for the project about $230.

RECAP

The data I was monitoring and displaying came from an old custom telephone switch that joined a large number of handset circuits onto any number of conference calls. The callers could pass information to each other simultaneously.

However, without a way to monitor which caller was on which conference, some users were concerned about security. Anonymity encouraged malcontents to occasionally yell expletives on certain conferences. Basically, I was implementing Caller ID on our system. The handset-site locations are given in Listing 1.

The switch was microprocessor based. Earlier, I found the spot on the backplane where the CPU passed in-

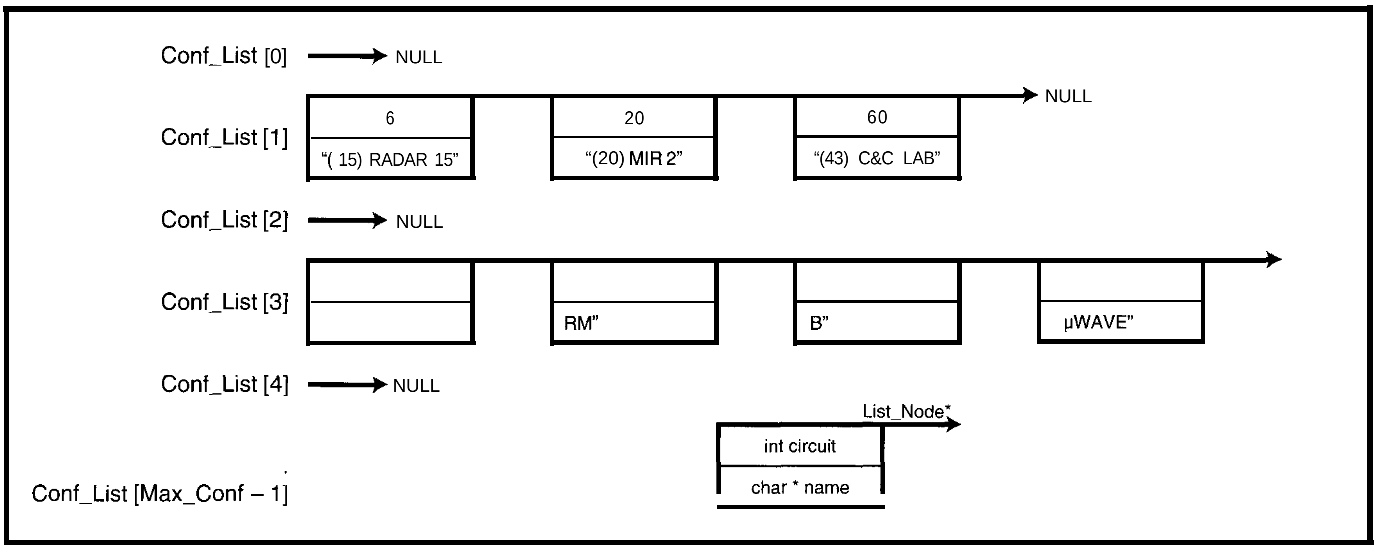


Figure 1—*The* structure of *the* array of *linked lists makes if easy to build up, modify, and break down groups of phones in conference calls.*

formation about who was on which conference to the other system components. I pulled out my handy-dandy Hewlett-Packard 4952A protocol analyzer and monitored enough traffic to figure out what was going on.

The HP-4952A is probably the most useful piece of test equipment I have. It displays, stores, and simulates multiple serial data protocols with RS-232, RS-422/-449, and V.35 requirements with its external interface pods.

If it worked at data rates above 64 kbps, I'd probably marry it. Sure, it has its faults. Like all HP equipment, it costs too much. Still, you can pick up a used one for less than half price at GE Rental or Lease, RAG Electronics, or any company selling used test equipment.

If you are not thrifty, but deep down stingy, try anonymous ftp to <ftp://oak.oakland.edu/pub/simtelnet/msdos/io_util>. The programs comchkxx.zipanddlmxxx.zip turn your PC into an asynchronous-data-line monitor. While not the same as an HP-4952, sometimes being thrifty means not being too picky.

REPRISE

Flush with my knowledge of the system, I started cranking out code. I chose an array of linked lists with one linked-list data structure for each conference. I'd add and remove nodes representing handset circuits from the appropriate conference linked lists as circuits joined, exited, or switched conferences. When a circuit's status changed, I'd write it to the screen.

It took a while to shake the cobwebs loose, but it wasn't long before I whipped some code out. Luckily, I didn't need a full-blown linked-list implementation. I wrote member functions to append a new caller, remove a caller, and print out the callers on the linked list (see Listing 2).

Listing 2—*The C++ linked-list class definition for the linked-list c/ass I wrote for this program contains the bare minimum linked-list features /need.*

```
class List-Node {
// Only the List class should be allowed to manipulate the
// list nodes. So, make everything private, but allow the
// List class access by making it a friend.
  private:

  friend class List;

// A list node consists of a pointer to the next node, the
// circuit number, and a pointer to the name in the location.
  List_Node(int c, char * n){
    next=NULL:
    circuit=c;
    name=n;

  ~List_Node(){}

  List-Node * next;
  int  circuit:
  char * name:
};

class List {
  public:

  ListO {
    head=tail=NULL;


  ~List();
  int head_delete(void);
  int append(int, char *);
  int remove(int);
  int print_list(int);
  int list_has_nodes(void);

  private:
  List-Node * head, * tail;
};

class  Array_of_Lists{
  public:
  Array_of_Lists(int size) {
    lp=new Listlsizel;
  }
  ~Array_of_Lists() {
    delete II lp;

  iist & operator[](int);
  int print_array();
  int remove(int);

  private:
  List * lp;
};
```

Figure 1 shows how the strings in Listing 1 are held in the linked list.

The HP-4952A let me simulate the system, so I worked out the bugs fairly quickly. Thinking back, if I'd done an anonymous ftp to <ftp://oak.oakland.edu/pub/simtelnet/msdos/cplusp1s> and grabbed some linked-list routines, I'd have saved about a week of coding and debugging.

The main routine was a different story. Using the C Async Manager, I found a ardtrm_a2() routine that read in data until its fixed-length buffer was full or it encountered a user-defined terminating character.

I set up my program to give me a buffer full of data whenever an end-of-message byte came from the system. Then, I used the strchr() function from the string.h library to find the end-of-message byte in the buffer.

I'd read a few of the previous bytes to see which caller was being added to or removed from which conference. I'd update the linked lists and print the data onscreen. Voilà, instant Caller ID!

I remember going to lunch with the other engineers and saying perhaps I'd been too harsh when I ripped the PC's design to shreds the other day. Ican be rather magnanimous when I'm about to complete a project in record time.

My head swelled to enormous proportions as I hooked up the PC to the switch and observed data appearing onscreen. There were bugs to swat, but everything was on schedule.

Of course, things were a bit more complicated. I had to deal with a dreaded boundary condition. Suppose that just before my buffer reached maximum length, the first few message bytes came across, filling the buffer to capacity.

The buffer didn't contain my end-of-message byte, which was at the start of the next buffer. So, I had to double buffer-that is, to maintain two buffers to guard against my message being split and lost.

If my message was cut in two, I read the last few bytes from the previous buffer to reconstruct it. My code became a bit messy taking care of this contingency, but it worked fine.

RETRENCHMENT

One morning, disaster struck. Although the evening before, things were going smoothly, everything stopped working. What had happened?

I stamped around for a while checking cables and connectors. I gave the PC a few three-fingered salutes (i.e., Ctrl-Alt-Del), all to no effect.

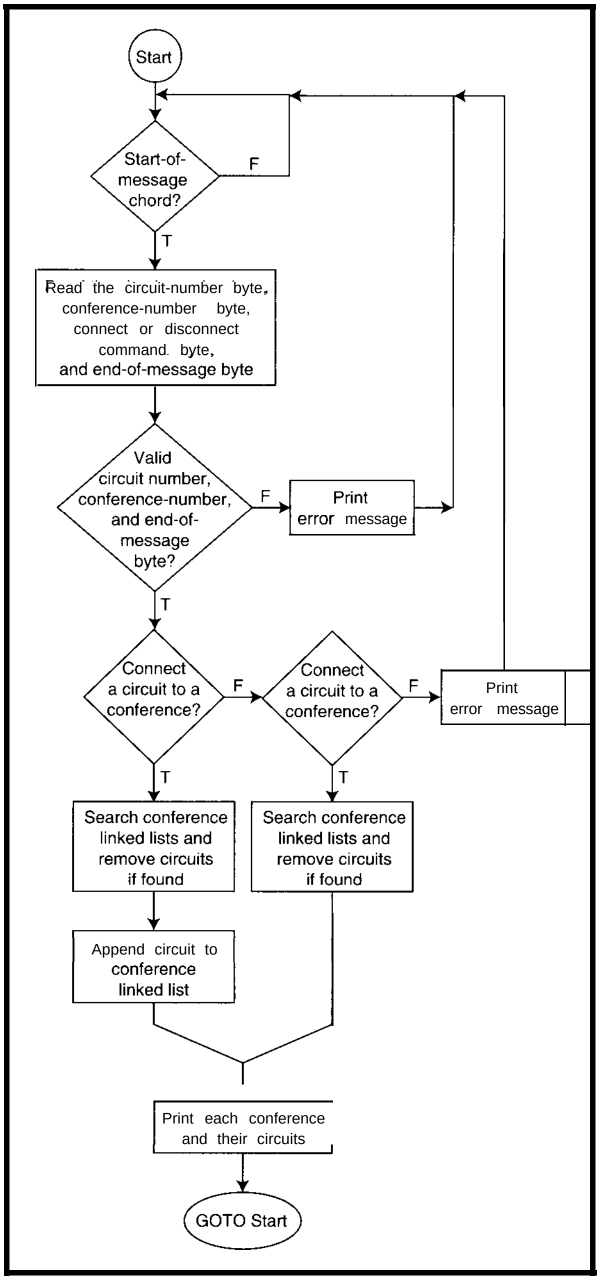
Then, I hauled out the HP-4952A to see if I was still getting data. I was— but it had changed! The end-of-message byte was now $AB, not $AA.

While I was beating my head on the wall, one of the technicians who maintains the equipment wandered by to make sure I wasn't banging anything he'd have to fix. After I explained the situation, he said the end-of-message byte probably changed because the switch was a redundant system.

The switch was composed of two identical switching sections. If an error was detected in one, it automatically changed to the other. He surmised that the end-of-message byte indicated which switch was the operational one at the time.

To test his theory, he manually changed the system to use its other section. Sure enough, the end-of-mes-

Figure 2-A *flowchart can be useful for figuring out how the* **different** *strings in Listing* **2** *relate to one* **another.**


Start
Start-of-message chord?
F
T
Read the circuit-number byte, conference-number byte, connect or disconnect command byte, and end-of-message byte
Valid circuit number, conference-number, and end-of-message byte?
F
Print error message
T
Connect a circuit to a conference?
F
Connect a circuit to a conference?
F
Print error message
T
T
Search conference linked lists and remove circuits if found
Search conference linked lists and remove circuits if found
Append circuit to conference linked list
Print each conference and their circuits
GOTO Start

sage byte changed back to a $AA.

Now I **was in real** trouble because of my choice of a PC for this project. If I'd written all the code myself in assembler, I could just have modified it to look for a $AA or $AB end-of-message byte.

But, I was using those canned routines that only allowed you to look for one byte. My head filled with devious workarounds. If I just cut a trace on the motherboard here and.. . In reality, my program's logic was doomed.

REPRIEVE

I think about work when I run. My friends say it's because anyone running that slowly has plenty of time to pontificate between each step. As I pounded the pavement, I concentrated on the problem of the multiple end-of-message bytes.

Well, there are two ways to deal with any problem-either do it or don't. The solution here was not to deal with the end-of-message bytes but with the start-of-message byte.

The start-of-message byte didn't change on the messages. So, I decided to look for it instead. I set up the rdtrm_a2() routine to read in characters until a start-of-message byte was encountered and then read in the next few bytes of the message.

No more strchr() routine, no more boundary conditions and double buffers. Why didn't I do this in the first place?

Perhaps the rdtrm_a2() routine made me feel obligated to read in a terminal instead of a start-of-message character. As shown in Figure 2, this solution reduced my program size and simplified my logic immensely.

RELIABILITY

This PC's reliability did concern me. It needed to run 24 hours a day, seven days a week. It was no ordinary PC sitting on a desk, only used during business hours. I attacked this problem by trying to avoid the frailties that most often cause PCs to malfunction.

Certain PCs seem to go through the batteries that back up their CMOS memory-contained setups at an unusually high rate. If you haven't had the pure joy of trying to figure out what disk drives (and their corresponding drive-type numbers) are on a PC with a dead battery, you haven't lived yet.

If you can't stand the excitement, buy a rechargeable battery instead of standard lithium batteries. I bought the Batpac from Industrial Computer Source.

These batteries take power from a disk-drive power connector to continuously recharge the backup battery. In case you don't have a spare power connector, they provide another connector so you can daisy chain the power on to your disk drive.

However, I suggest you grab a magic marker and write the drive-type number on the disk itself. Do it now—you'll thank me later.

Another component somewhat prone to failure is the hard-disk drive. Hard disks have come a long way, but I'll wager that something that doesn't spin at 2000 rpm lasts longer than something that does.

I spent some time researching solid-state disk drives and putting my program into EPROM or flash memory. I finally decided to go with something extra thrifty-a floppy-disk drive.

I yanked out the hard drive, made a bootable floppy disk, copied my program to the floppy, and had the AUTOEXEC.BAT file run my program from the floppy disk at bootup. Sure, a floppy disk spins, but since my system is rarely rebooted, the disk is seldom accessed.

Of course, you can only do this for a program that fits on a floppy disk and doesn't write much of anything to it. But, these are the types of simple embedded designs I am targeting.

To be legal, you must purchase a copy of MS-DOS for each PC running MS-DOS. A thrifty solution is to use one of your old copies of MS-DOS (or PC-DOS, DR-DOS, etc.) laying around.

Anyone can scrape up an unused copy of DOS V.3.3, DOS V.4.0 (yikes!), or DOS V.5.0. Most of the features added to DOS over the past few years aren't needed by an embedded system.

One last unrelated hardware problem arose from using a PC as a stand-alone system-the keyboard. Nobody wants a keyboard around, cord in a tangle, falling off shelves, tempting the button pusher in all of us to try a few keys.

Fortunately, it wasn't a real problem. My BIOS setup let me ignore keyboard errors and thus operate without a keyboard.

Most newer BIOSs have this feature, but since this article is about getting some use out of forgotten PCs, here's a solution. Vetra Systems produces a little box called Eliminator that plugs in the keyboard port in place of a true keyboard, fooling a system into thinking one is present.

RELAX

The little-AT-that-could has been in operation for over a year with no difficulties. I socked away a spare CGA card and an interface card, so in the unlikely event of a catastrophic failure, we can drag a PC off a desk and be up and running again in a matter of minutes.

Since most of our technicians are familiar with PCs, I won't necessarily receive an emergency call to come in and direct resuscitation efforts if something fails. All in all, I'm pleased with the outcome. In the proper situation, with the proper precautions, PCs can become part of an effective embedded system. ▣

Daniel Nygren received his BSEE from Clemson University and his MSEE from the University of Texas at Austin. His interests include real-time fault tolerance, aircraft command and control systems, and tactical digital information links. You may reach Dan at nygren@tecnet1.jcte.jcs.mil.

SOFTWARE

Files are available via anonymous ftp from <ftp://oak.oakland.edu/pub/simtelnet/msdos/turbo_c/>. The files asyncnch.zip, ibmcom_c.zip, and tcommxx.zip are similar routines. The libraries comlibxx.zip and com_int.zip are also routines to try. Files are also available from the Circuit Cellar BBS and Software on Disk for this issue. See ConnecTime in this issue for more information.

SOURCES

MAX233 Dual RS-232 transmitters and receivers chips
Maxim Integrated Products
120 San Gabriel Dr.
Sunnyvale, CA 94086
(408) 737-7600
Fax: (408) 737-7194

AM26LS31 and AM26LS32 quad RS-422 line drivers and receivers
Advanced Micro Devices
P.O. Box 3453
Sunnyvale, CA
(408) 732-2400

HP4952A protocol analyzer
Hewlett-Packard
Portable Computer Division
1000 NE Circle Blvd.
Corvallis, OR 97330
(503) 75 7-2000

C **Async Manager**
Blaise Computing, Inc.
819 Bancroft Way
Berkeley, CA 94710
(215) 443-9705
Fax: (215) 886-6707

RW-100TS
Quatech, Inc.
662 Wolf Ledges Pkwy.
Akron, OH 44311
(216) 434-3154
Fax: (216) 434-1409

Turbo C++
Borland International Inc.
1800 Green Hills Rd.
Scotts **Valley, CA 95067**
(408) 438-5300

Eliminator
Vetra Systems Corp.
275-J Marcus Blvd.
Hauppauge, NY 11787
(516) 434-3185
Fax: (516) 434-3516

Batpac
Industrial Computer Source
P.O. Box 910557
San Diego, CA 92191-0557

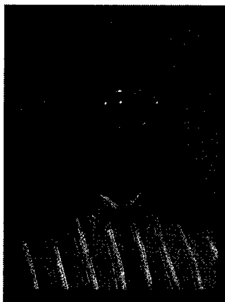
IRS

419 Very Useful
420 Moderately Useful
421 Not Useful

DEPARTMENTS

Tools

Part 1: Schematic Capture—the Proverbial Electronic Napkin



Schematic capture programs have three main components—the parts library, drawing editor, and postprocessor. Jeff shows us how to use these to guestimate cost and guide the PCB layout artist.

FROM THE BENCH

Jeff Bachiochi

I chuckle when I hear about the paperless workplace. We use more paper today than we ever did.

From grocery-store receipts, ticket stubs, and the morning paper to bank statements, office memos, and sales reports-it's extremely difficult to give up the security of a physical copy of each and every transaction.

It's not that we haven't made progress. Computers store more information. But, it's comfortable holding paper instead of a piece of magnetic medium which, although it contains more information and is written in a global language, can't be directly interpreted by a human.

I've come to appreciate the benefit of a word processor over typing reports, correspondence, or a monthly column. The ability to create, edit, store, recall, view, and print saves hours of unnecessary typing. At my hunt-and-peck typing rate, those hours add up. The first time I needed to change some correspondence, I saw the light.

In engineering, we all have stories of creating the schematic for a new product or solution on a dinner napkin. But, just imagine opening a user manual only to find a copy of the napkin reproduced complete with food stains. To avoid this kind of embarrassment, schematics of yesteryear were painstakingly redrawn using a full-size drafting board.

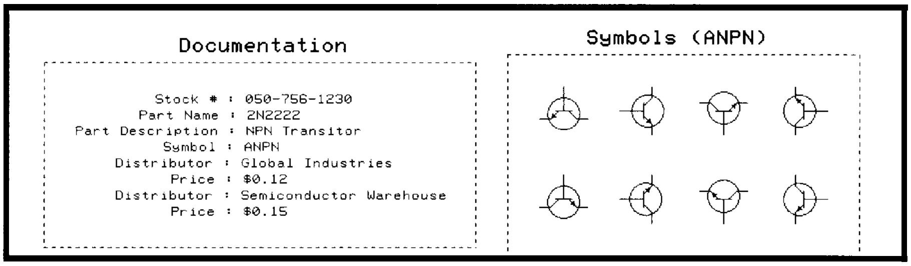
CAD

Such labor-intensive work has been unnecessary for many years now, thanks to a wide selection of schematic-capture packages. The best part is that you don't have to spend big bucks to get professional results.

Although these programs differ in look and feel, y need to know the three basic sections-parts library, drawing editor, and postprocessor-to use the programs effectively.

The parts library is a database of shapes or symbols and associated part information. The drawing editor displays and handles the parts' symbols and their interconnections. The post-processor creates documentation from the drawings and database.

Let's look at each of these in detail.



```
Documentation

          Stock # : 050-756-1230
        Part Name : 2N2222
 Part Description : NPN Transitor
           Symbol : ANPN
      Distributor : Global Industries
            Price : $0.12
      Distributor : Semiconductor Warehouse
            Price : $0.15
```

Figure l-Although many views of a part might be used, they *all* share the same *documentation*

PARTS LIBRARY

Just as books are stored on shelves in a public library, all electronic parts are stored in a repository of equal proportion. The number of predefined parts in the parts library varies based on the cost of the package.

The higher-priced packages might include whole libraries from individual manufacturers. But, even minimal libraries have many standard parts.

Each part in a library is made up of at least two pieces-the documentation and the symbol. Figure 1 shows the documentation and symbols one might have for a transistor.

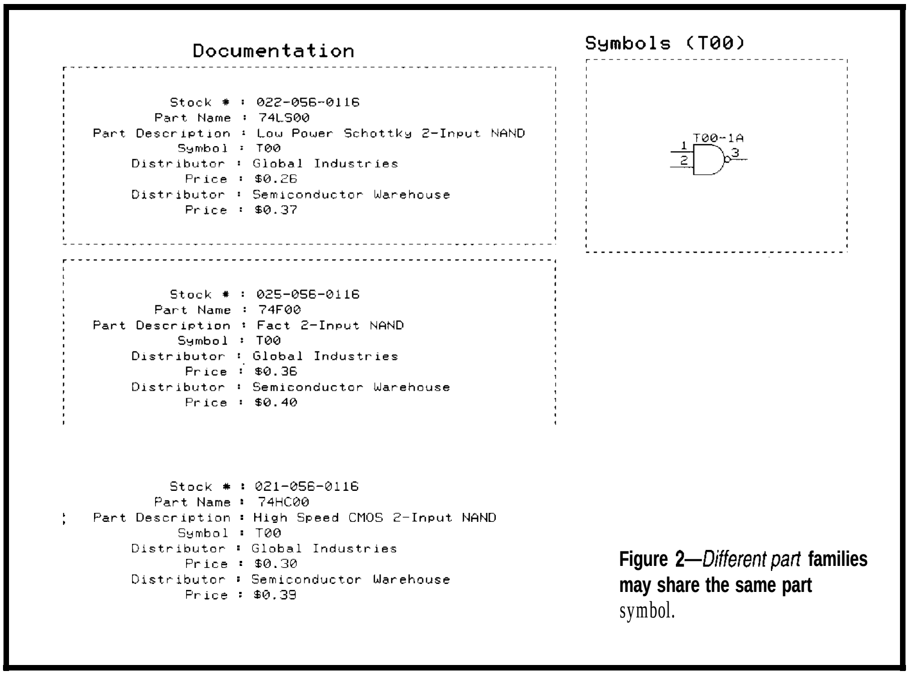
The documentation contains information describing the device. It might include a device name and stock number, where it can be purchased, and which symbol the drawing editor uses to define its physical appearance on the schematic page.

As shown in Figure 1, several orientations of the same part are available. Parts individually defined in the documentation may share the same symbol (e.g., 74LSO0 and 74F00}. Figure 2 shows how parts can have different documentation but the same symbol.

The parts symbol stored in the library reflects the part's appearance on the page and the electrical information associated with each pin of the device. This information describes the function of each device pin and is used in postprocessing for error checking.

Associated with each pin is a point on the symbol where any external connection must be made to connect to the device. This point is usually the end point of what it would look like on the symbol's component pin.

To manage the library's size, parts can have generic names like "resistor," where the part's specifics are entered when it is used in a drawing.



```
Documentation

          Stock # : 022-056-0116
        Part Name : 74LS00
 Part Description : Low Power Schottky 2-Input NAND
           Symbol : T00
      Distributor : Global Industries
            Price : $0.26
      Distributor : Semiconductor Warehouse
            Price : $0.37

          Stock # : 025-056-0116
        Part Name : 74F00
 Part Description : Fact 2-Input NAND
           Symbol : T00
      Distributor : Global Industries
            Price : $0.36
      Distributor : Semiconductor Warehouse
            Price : $0.40

          Stock # : 021-056-0116
        Part Name : 74HC00
 Part Description : High Speed CMOS 2-Input NAND
           Symbol : T00
      Distributor : Global Industries
            Price : $0.30
      Distributor : Semiconductor Warehouse
            Price : $0.39
```

Figure 2—*Different part* families may share the same part symbol.

Symbols, too, take on specific information when used. Both the part type (e.g., 74LSOO) and designation (e.g., U1} are attached during placement.

You can see there's more to the schematic-capture library than just a bunch of pictures. The library database may be divided into many smaller libraries sorted by device type (e.g., linear, digital, memory, processors, etc.] or manufacturers (e.g., Motorola, National, Siemens, etc.)

DRAWING EDITOR

Substitute the computer screen for a dinner napkin and use the drawing editor to place parts on a page. To do this, the part name is entered and the parts library is searched.

If a match is found, the symbol for that part is placed on the page. You can move the symbol around the page and choose the necessary orientation.

Parts should be placed in a logical fashion. Generally, circuit flow moves left to right (i.e., inputs on the left and outputs on the right). Or, start with the largest part first (e.g., a processor] and place the smaller ones around it.

Large circuits that don't fit comfortably on a single page can be broken down into separate pages by function. Don't crowd too many parts too close together. Think of the circuitry as a block diagram. Place the parts which make up each block on a separate page.

But, parts alone do not a schematic make. Connections between parts create a circuit's character. All connections in a single path make up a net.

A net is created when you connect two or more part symbols' pins together by drawing a line (i.e., wire) between them. For identification, each net is given a name either by the user or automatically by the program.

It's almost impossible to draw a schematic without crossing any lines.

Crossing lines are considered connected when a dot (which looks like a blob of solder] is placed on the point of intersection.

To eliminate the possibility of misunderstanding when lines are crossing or connected, follow this simple rule: a connection is always shown as a "T", and two lines crossing is an "X".

When many connections must be made between parts, the web of individual lines becomes confusing. A bus connection simplifies the page.

Think of each connection or line as a wire and a bus as a multiconductor jacketed wire (drawn as a wide line). Each individual wire can enter or exit the jacket at any time. When it does, it must be identified.

In reality, color coding each wire identifies it. On the schematic page, labels are used. Colors can be labels, but signal names make more sense.

Schematic pages with a single drawing name are all part of the same circuit, and any pages with the same net label are connected. Just as too many individual lines can be confusing on a schematic, a bus containing all connections can be equally confusing.

I use multiple buses. For example, when drawing a microprocessor, I like to use three buses-an address bus, a data bus, and a control bus. Since the signals within each bus are related, it's easier to visualize the flow.

A good drawing editor not only functions as a schematic drawing and editing tool, but it also creates and maintains your parts library. Every time I draw a new schematic, there's at least one new part. The editor lets me easily document the new part and then define a symbol by drawing it.

You can design the symbol using the normal left-to-right I/O flow or be creative. You can draw the symbol the way the part physically appears (e.g., a PLCC chip pinout, as shown in Figure 3). Doing this gives the technician probing the PLCC socket a visual cue to where each signal can be found.

After the symbol is drawn, its pins (i.e., connections) must be defined. You must define external connections to the part and the internal functions of the pin within the device. The pin might be classified as an input, output (i.e., open collector or tristate), bidirectional, or don't-care.

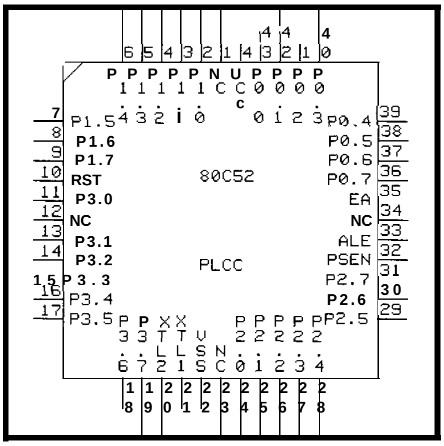


Figure 3-The processor is shown as it *really* looks, making it easy to troubleshoot *with* a scope probe.

It's advantageous to create special versions of part symbols for a particular drawing. Programmable devices can to be customized according to use.

Using positive and negative logic forms clarifies a circuit's function. All logic can be described using both positive and negative as shown in Figure 4.

Positive logic describes the logic function based on the high logic level of the input, as in **"1** AND **1"** (equals **1).** The same AND gate described in negative logic is "0 NOR 0" (equals **1).** When describing the control logic of a logic-low chip select, negative logic improves clarity.

POSTPROCESSING

If you use a schematic for documentation only, you may not need to use postprocessing. However, it points out drawing errors, produces a bill of materials, and creates links to other CAD programs.

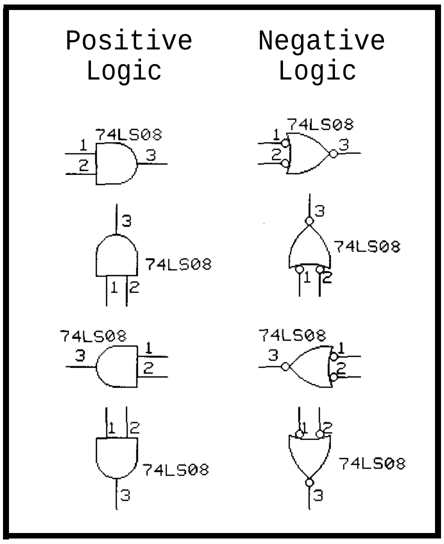


Figure 4—*Positive logic* **describes** *a* **function** *using "high" levels,* **while** *negative logic describes a function based on "low" level input.*

The postprocessor uses the information contained in the parts symbol to detect potential errors. Each net is analyzed to determine whether certain rules have been broken.

The error report points out nets with no driving source (i.e., floating), more than one permanent driving source, single pins with labels, and nets with multiple labels. The postprocessor also points out parts with the same parts designator or that use a gate within a package more than once.

Naturally, the postprocessor can't locate a logic error. That's left up to your engineering skill. (You can always turn to a timing analysis program to debug the logic.)

To actually build the circuit you have drawn, you'll need a bill of materials (BOM). The postprocessor lists all the parts used in your drawing.

Quantities, part names, stock numbers, manufacturers, distributors, pricing, and the like can be pulled from the parts library's documentation section. Most BOMs can be customized to your own parameter format.

To physically wire a prototype, the postprocessor can prepare a wire list. The wire list is a human-readable list of nets. Wire wrapping-popular for many years-used the wire list to daisy chain all the connections within a net. The list is also good for "ohming out" a finished prototype to ensure no connections are left out.

A postprocessor can also produce a machine-readable list of nets. The net list is all a PCB program needs to ensure connectivity in creating a PCB layout.

The catch is that every PCB-layout program requires a net list in its own format and every schematic drawing package creates its own net list format. You'd think these companies could agree on a standard format, but most won't accept another format directly.

Fortunately, many packages include pin/net conversion programs to convert their format to one used by other PCB-layout packages. Companies with a complete line of CAD aids don't usually require conversion between packages.

If you're using one company's schematic-drawing package and another's PCB-layout package, make sure a conversion program is available.

THAT'S NOT ALL

Since these programs enable you to draw lines and generate text, the schematic-capture program can be used to create block diagrams and flowcharts. Although you can't do timing analysis, you can draw timing diagrams.

Many libraries contain physical symbols of parts in relation to other parts. You can experiment with parts placement on a pseudo-PCB drawing which guides the PCB-layout artist.

A lot of information cannot be transferred in a simple net list **(e.g.,** the importance of placing the decoupling capacitors near the ICs). The net list only indicates that decoupling capacitors and ICs are connected in parallel.

Once a schematic is drawn, I'm usually asked how tightly I can cram these parts together. Using the schematic package for parts placement offers an early estimate of board space and passes placement information on to the PCB-layout engineer.

CHECK, PLEASE

Next time you're at lunch, save the napkins for the spaghetti sauce on your shirt. And when you're back at the office, get comfortable with your schematic-capture package.

Tune in next time for the basics of PCB layout and what's necessary to get that circuit fabricated. ■

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

SOURCES

Tango, PCAD
ACCEL Technologies, Inc.
6825 Flanders Dr.
San Diego, CA 92121
(619) 554-1000
Fax: (619) 554-1019

Eagle
CADSoft Computer, Inc.
801 S. Federal Hwy.
Delray Beach, FL 33483
(561) 274-8355
Fax: (561) 274-8218

PCBoards
MicroSim Corp.
20 Fairbanks
Irvine, CA 926 18
(714) 770-3022
Fax: (714) 455-0554

Harmony
OHIO Design Automation, Inc.
Millyard Technology Park
78 Pine St. Ext.
Nashua, NH 03060
(603) 598-2525
Fax: (603) 598-2626

OrCAD
OrCAD
9300 SW Nimbus Ave.
Beaverton, OR 97008
(503) 671-9500
Fax: (503) 671-9501

PowerPCB
PADS Software, Inc.
165 Forest St.
Marlboro, MA 01752
(508) 485-4300
Fax: (508) 485-7171

Protel EDA
Protel Technology
4675 Stevens Creek Blvd., Ste. 200
Santa Clara, CA 9505 1
(408) 243-8143
Fax: (408) 243-8544

GerbTool
Wise Software Solutions
8285 SW Nimbus Ave., Ste. 191
Beaverton, OR 97008
(503) 626-7800
Fax: (503) 520-I 759

CadStar
Zuken-Redac, Inc.
238 Littleton Rd., Ste. 100
Westford, MA 01886
(508) 692-4900
Fax: (508) 692-4725

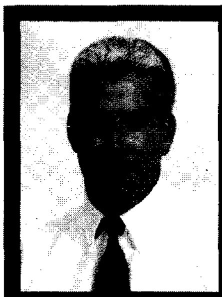
IRS

422 Very Useful
423 Moderately Useful
424 Not Useful

Flash Fight Flares

SILICON UPDATE

Tom Cantrell



When it comes to flash, Intel and SanDisk are warmed up and in the ring. The big question: what factors will decide this match—speed, weight, size, techniques, trainers, fan support.... Where will you place your bet?

Press releases proclaim a consortium of industry visionaries promising to deliver new technology. The fax then shifts into overdrive as retaliatory PR from another hastily formed alliance touts a similar, but incompatible, solution.

For Joe or Jane Blow, high-tech widgeteer, it's tempting to don a flak jacket and duck down. But, that's not an option in a business where you're quick or you're dead.

Of course, choosing wrong can be career limiting as well. But, at least you go down in flames rather than being meekly led to slaughter.

Technical evaluation of the alternatives is necessary-but not sufficient—for choosing a winner. High tech has lots of better mousetraps that failed for marketing and business reasons.

Peruse each camp's team. A competitor is in group A, so join to keep an eye on them. Oops, your biggest competitor or favorite supplier belongs to *both* teams!

And, here they go again. In one corner, we have CompactFlash (CF), coached by SanDisk, and in the other, the Miniature Card (MC), backed by Intel. Will it be a quick knockout or a split decision?

HIGH STAKES

As shown in Photos 1 and 2, respectively, the CF and MC are both tiny cards that pack large amounts of flash (or other) memory into matchbook-sized packages.

Besides obvious applications such as PDAs and portable data loggers, interest is fueled by the emergence of digital cameras and digital film. That's why the teams include names like Kodak, FujiFilm, Canon, and Polaroid [see sidebar "Team Rosters").

In many ways-thankfully, not too many-CF and MC can be considered downsized PCMCIA cards. Indeed, both offer PCMCIA adapter gizmos so new cards can plug into existing slots.

It's no secret I've been something of a PCMCIA naysayer from way back. What started as a simple memory card suffered badly from creeping specitis.

The profusion of Types (I, II, and III), some wild and crazy connector schemes, and a dubious mishmash of OS and driver software gave a new meaning to plug-and-pray. Nevertheless, PCMCIA can claim some success, especially in notebook PCs.

Presumably, the new standards won't suffer the same fate. That they came up with names you can say gives hope that the people in charge are on top of things.

Sharing basic form factor, market aspirations, and PCMCIA-retrofit path, CF and MC are otherwise quite different under the hood. Let's take a look.

IDE DISK, THEREFORE I SPIN?

The CF pinout shown in Table 1 may look familiar since it's similar to the 68-pin PCMCIA predecessor. As before, CF makes the flash look like an IDE disk drive.

The beauty of such a scheme is that all PCs, and indeed most other platforms like Macs and workstations, already have software drivers built in or easily available.

Photo 1—*The CF* **card packs megabytes of flash into a** *matchbook-sized package. Like its PCMCIA predecessor, the card emulates an IDE disk drive.*

Photo 2—*The* MC is effective/y a big memory chip that *accommodates SRAM and DRAM as well as flash.*

On the downside, the on-card controller (a 68k-based custom chip from Motorola) adds to the CF card's cost. The amount and relative importance of the adder is a point of contention.

Though some may argue whether or not making the flash look like a disk drive is elegant, there's no doubt that adding intelligence on the card makes life easier for the host.

Most activity centers on a couple dozen pins, including the 8- or 16-bit data bus, some address lines, and basic control signals such as *OE, *WE, *IORD, and *IOWR.

Actually, the CF card can be configured at *RESET in a number of modes, such as primary, secondary, contiguous, or memory-mapped I/O and so on. There's even a "True IDE" mode that changes the function of a few I/O pins to mimic the disk interface electrically and in software.

Once you cut past the gobbledygook, they're simply different ways, largely relics of past PC hacks, of routing commands and data into the card.

Primary and secondary modes place the CF IDE surrogate registers at the same I/O addresses traditionally allocated to disk drives (1 F0h and 170h, respectively). However, to preserve these I/O addresses for real disks, contiguous-I/O mode lets registers be mapped to other I/O addresses. All the I/O modes rely on CPU In/Out instructions (i.e., using the IORD and IOWR pins) for access.

By contrast, the memory-mapped mode allocates the card to the PC memory space, so access relies on the *OE (read) and *WE (write) pins. In a clever memory-window feature, references to incrementing addresses are funneled across the on-card FIFO. This offers handy instructions like **REP MOVSB** (byte)or **REP MOVSW** (word).

Both modes rely on *CE1 and *CE2 to access low (DO-D7) and high (D8–D15) bytes of the data bus, respectively. For an 8-bit host, only *CE1 is used, and D8–D15 (and the host) connect to DO-D7.

In the old days, flash technology required a separate, higher-voltage (e.g., **12** V) power supply for writing. But, that hassle went away with the appearance of single-voltage flash chips.

The bad news is, with the quest for lower power, the voltage keeps shrinking. The VS 1 and VS2 (voltage sense) outputs let the card request 5 V, 3.3 V, or an even lower unspecified voltage.

The *REG pin is a kind of address line that chooses between various card-configuration and control registers and the IDE register map. The former are normally accessed at card insertion to determine card capabilities and to configure it in the proper mode. Subsequently, flash data transfers rely on IDE registers (see Table 2).

For those who aren't experts about PC disk stuff-don't worry, I'm not either. It seems a rather simple matter of setting up the sector (5 12 bytes) address (head, cylinder, sector), issuing a command, transferring the data, and monitoring the status.

The command list includes all the disk-related stuff for compatibility, though many of the commands (e.g., Recalibrate, Seek) have little meaning for flash. Even the Wear Level command needed for earlier flash cards is ignored thanks to the 300,000-cycle write-endurance spec. Ultimately, almost all the action centers on the basic read and write sector.

Figure 1 shows the register setup for read sector. The card can be configured for Logical Block Addressing, which substitutes a simple linear translation (i.e., Block O-n) for the somewhat archaic head, cylinder, and sector specifiers. Both ways point to a single 512-byte block of flash.

Issuing the command causes the card to go busy (status bit and pin), get

Number	Name	Type	Number	Name	Type
1	GND		26	*CD1	O
2	D03	I/O	27	D11[1]	I/O
3	DO4	I/O	28	D12[1]	I/O
4	D05	I/O	29	D13[1]	I/O
5	DO6	I/O	30	D14[1]	I/O
6	D07	I/O	31	D15[1]	I/O
7	'CEI	I	32	*CE2[1]	I
8	A10	I	33	*VS1	O
9	*OE	I	34	*IORD	I
10	A09	I	35	*IOWR	I
11	A08	I	36	*WE	I
12	A07	I	37	RDY/BSY\IREQ	O
13	v c c		38	VCC	
14	A06	I	39	*CSEL	I
15	A05	I	4 0	*VS2	O
16	A04	I	4 1	RESET	I
17	A03	I	42	*WAIT	O
18	A02	I	43	*INPACK	O
19	A01	I	44	*REG	I
20	A00	I	45	BVD2	I/O
21	D00	I/O	46	BVD1	I/O
22	D01	I/O	47	D08[1]	I/O
23	D02	I/O	48	D09[1]	I/O
24	WP/*IOIS16	0	49	D10[1]	I/O
25	*CD2	0	50	GND	

[1] These signals are needed for 16-bit access and are not in 8-bit systems. Dual-purpose signals show the the memory-only interface signal before the I/O interface signal. The *IORD, *IOWR, and *INPACK are not used in the memory-only Interface.

Table l--Though *reduced from 68 pins to 50, the CF shows its PCMCIA roots in the electrical and mechanical (i.e., pin and contact) interface.*

'REG	A9–A4	A3	A2	Al	A0	*IORD = 0	*IOWR = 0
0	1F(17)	0	0	0	0	Even RD Data	Even WR Data
0	1F(17)	0	0	0	1	Error Register	Features
0	1F(17)	0	0	1	0	Sector Count	Sector Count
0	1F(17)	0	0	1	1	Sector Number	Sector Number
0	1F(17)	0	1	0	0	Cylinder Low	Cylinder Low
0	1F(17)	0	1	0	1	Cylinder High	Cylinder High
0	1F(17)	0	1	1	0	Select Card/Head	Select Card/Head
0	1F(17)	0	1	1	1	Status	Command
0	3F(37)	0	1	1	0	Alt Status	Device Control
0	3F(37)	0	1	1	1	Drive Address	Reserved

Table 2—*To mimic a disk drive, the CF card presents an IDE-compatible register map to the host*

the sector from flash into the on-card FIFO, clear busy, and signal (status bit and interrupt) data availability. The host can read 512 bytes or 256 words of data from the FIFO.

For writes, notice w/o Erase. A write sector erases the data first—which wastes a lot of time. Instead, use erase sector and then write without erasing at high speed.

The CF on-card micro handles ECC, bad-block mapping, dynamic adjustment of performance versus power consumption, extensive self-testing and diagnostics, and the gory details of accessing the flash.

MINIWEIGHT CONTENDER

The main visible difference for the MC is its elastomeric connector, which is best described as a conductive rubber ribbon cable squished between the card and socket.

Not surprisingly, there are plenty of connector claims on both sides. The CF corner rejoins that the PCMCIA-like pin and socket they use is well-proven, and in fact, specs up to 10,000 insertions versus only 5000 for the MC.

The counterpunch is that the elastomeric connector is easy to replace and use. It tolerates real-world abuse.

To me, the connection-not connector-differences are much more profound. As shown in Table 3, the MC carries a full (AO-A24) address bus. Not a big deal, except that it indicates there's no on-card intelligence. The MC is thus like a big memory chip.

While most of the action centers around flash, the direct access (via ● CE, *OE, *WE, etc.) of the MC makes it amenable to other technologies like ROM, SRAM, and even DRAM. (Notice the ● RAS and ● CAS signals.)

There's a little more to explain. The *BUSY line handles the timing of flash-memory writes, but it can simply be grounded for a full-speed memory. *BS8 is an input to the card selecting 8- (DO-D7) or 16-bit (D0–D15) operation.

*CINS (card insert) is a grounded output on the front of the card that makes contact before the rest of the interface. *CD (card detect) is also a grounded output but grouped with the other signals. Thus, *CINS (plus V_{CC} and GND) give the host early warning of the card's arrival.

Similar to CF, *VS1 and *VS2, two voltage-sense outputs, request different voltages (e.g., 5 V, 3.3 V, and a future lower one). The MC also adds keying of the card and socket to prevent flameout (see Figure 2).

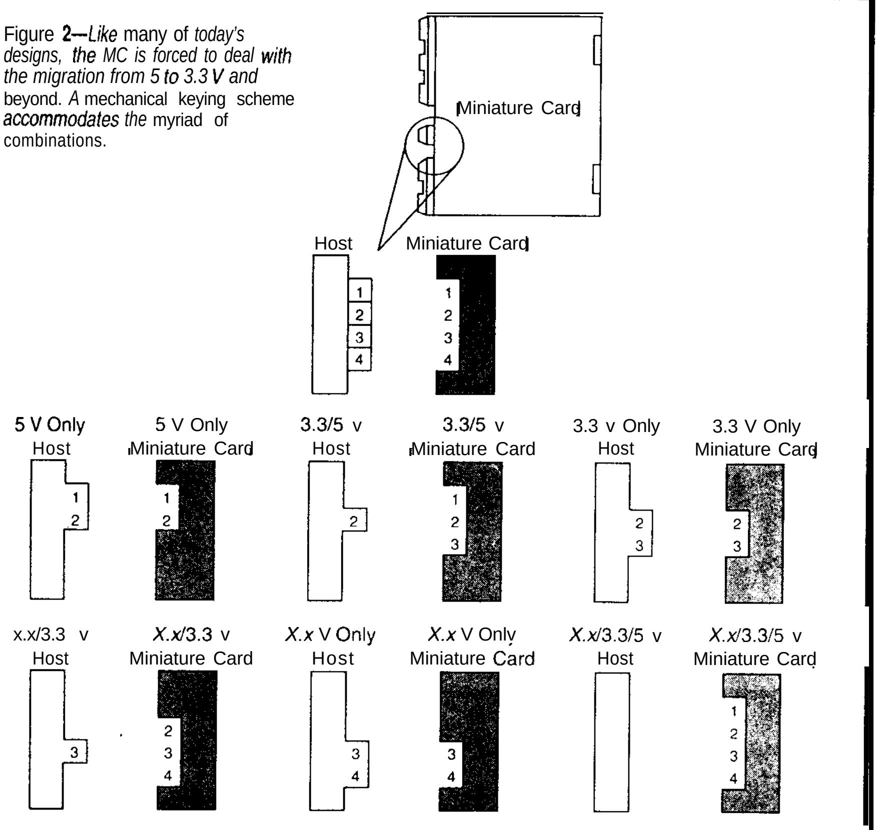
Otherwise, the only difference from a memory chip is that the MC includes an information block at the beginning for what's called the Attribute Information Structure (AIS) (see Table 4).

The AIS characterizes timing details and power requirements for up to four blocks of on-card memory. More information is provided for DRAMs and memory with burst capability.

Bit->	7	6	5	4	3	2	1	0
Comand (7)	20H + 21H							
C/D/H (6)	1	LBA	1	Drive	Head (LBA 27-24)			
Cyl High (5)	Cylinder High (LBA 23-16)							
Cyl Low (4)	Cylinder Low (LBA 15-8)							
Sec Num (3)	Sector Number (LBA 7-O)							
Sec Cnt (2)	Sector Count							
Feature (1)	X							

Figure 1—*Initiating access to a CF sector (512 bytes) simply requires specifying the address and sector count and issuing the command such as read, write, or erase.*

Figure 2—*Like many of today's designs, the MC is forced to deal with the migration from 5 to 3.3 V and beyond. A* mechanical keying scheme *accommodates the* myriad of combinations.



Since DRAM cards are usually volatile, storing the AIS is a problem. To solve it, the MC relies on the I^{2}C interface (using the SDA and SCL pins) to read the AIS from a serial (EEP)ROM. A mechanical key prevents DRAM cards from plugging into sockets that can't handle them.

Hosts that can handle all types, including DRAM, first try to read the AIS serially. If that doesn't work (i.e., AIS checksum fails), try a parallel read.

The card provides an optional write-protect switch which the spec implies should physically prevent writing (e.g., gate the WE' line). However, the spec also says the host should detect the protect status, without saying how.

SOFT TOUCH

Simplifying the card by removing the controller means host software does the housekeeping. Proponents argue that today's PCs have enough power to spare a controller's worth of CPU cycles in return for lower-price cards.

Rather than using a controller to fool the PC into thinking an MC is a disk, a layer of software known as a Flash Translation Layer (FTL) is sandwiched between the OS and hardware. This driver and BIOS extension soft-

Pad Number	Signal Name	Pad Number	Signal Name	Pad Number	Signal Name	Pad Number	Signal Name
1	A18	16	A23	31	A19	46	'CD
2	AI6	17	A22	32	AI7	47	A21
4	AI4	18	'OE	33	AI5	48	'BUSY
5	Vccr	19	D15	34	AI3	49	'WF
	'CEH	20	D13	35	AI2	50	D14
6	AII	21	D12	36	*RESET	:51	RFU
7	A9	22'	D10	37	A10	52	D11
8	A8	23	D9	38	• VS1	53	*VS2
9	A6	24	D0	39	A7	54	D8
10	A5	25	D2	40	*BS8	55	D1
11	A3	26	D4	41	A4	56	D3
12	A2	27	RFU	42	*CEL	57	D5
	A0	28	D7	43	A1	58	D6
13	*RAS	29	SDA	44	*CASL	59	RFU
15	A24	30	SCL	45	*CASH	60	A20

Table 3—*The electrical interface for MC looks more like a memory chip than a disk drive*

Team Rosters

Most of **the big** names in electronic systems including ICs, PCs, and digital cameras are choosing between the CF and MC standards, with many hedging their bets by supporting both.

CompactFlash Ass".
3M Corp.
AMP Inc
Apple Computer*
Asahi Optical Co (Pentax)
Canon Inc.*
Carry Computer Eng.
Centennial Technologies
Cirrus Logic
DataFab Systems Inc.
Eastman Kodak Co.'
Epson'
FujiFilm Microdevices
Fujitsu Computer Technology Ltd.
Fujitsu Takamisawa America
Geoworks
Gumball co.
Hirose Electric Inc.
Hitachi
Hewlett-Packard Co.*
Hosiden
Irvine Sensors Corp.
Kingmax Technology Inc
Kingston Technology Corp.
Konica
LG Semicon*
MCCI
Micron Quantum Devices
Mitsubishi
Mitsubishi Plastics Inc.
Molex
Motorola*
NEC'
Nikon
Norris Communications
Panasonic (Matsushita)*
Polaroid*
Pretec
Psion
Ricoh
SanDisk*
Sanyo
Seagate Technology*
Siemens Nixdorf
Simple Technology
SMART Modular Technology
Sony
TDK
ThinStar
Viking Components Inc.
(* founding companies)

Miniature Card Implementers Forum
Board Of Directors
Advanced Micro Devices
Compaq Computer Co.
Fujitsu Ltd.
Hewlett-Packard Co.
Intel Corp.
Konica Corp.
Microsoft Corp.
Olympus Optical Company. Ltd.
Nokia Mobile Phones Ltd.
Philips Semiconductor Inc.
Sharp Corp.

General Membership
Alps Co., Ltd.
Annabooks
Asahi Optical Co., Ltd. (Pentax)
Augat
Centennial Technologies Inc.
CDSM, Inc.
Duel Systems
Eastman Kodak
Elastomeric Technologies Inc.
General Magic
Hitachi Ltd.
Hyundai Electronics
Kingmax Micro Technologies
Micron Quantum Devices
Mitsubishi Electronics America, Inc
Mitsubishi Plastics Inc.
Moore Computer Consultants Inc.
Motorola
M-Systems
Nikon Corp.
NRC Systems, Inc.
PC Card Packaging
Phoenix Technologies
I'NY Electronics
Ricoh co., Ltd.
Sarnsung Semiconductor
Sanyo Electric, Co., Ltd.
SCM Microsystems
Simple Technology
Smart Modular
Sony Corp.
Sycom Technologies, Inc
Vulcan Enterprises

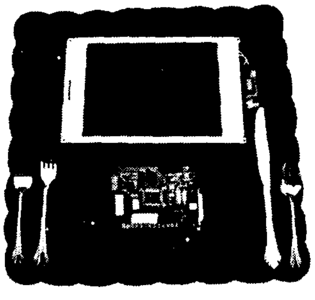
ware is what turns a disk command into the low-level flash cycles.

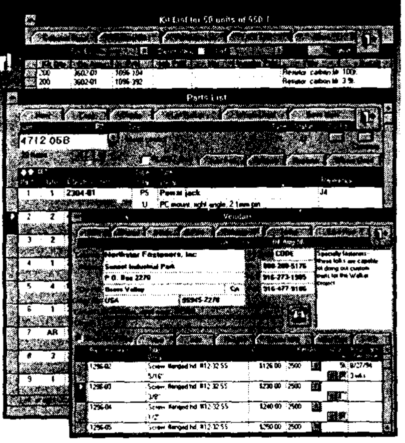
The downside of this and any software-driven scheme is the dreaded "driver disk" dilemma. Anytime host and/or memory technology changes, it's time to fumble for the right floppy.

They hope to standardize the FTL and form a Common Flash Interface specification. It's premature to judge the outcome, but the MC roster includes the necessary players.

Remember, most of the who's-in-charge (i.e., controller or host) controversy revolves around the vagaries of flash. MC cards based on other technologies (e.g., ROM and RAM) interface like any other memory chip.

Indeed, a subtle but possibly important advantage for MC is the so-called Execute-In-Place (XIP) capability. This





Address	Section	Description
OOH-OFH	PC Card Compatibility Area	Reserved for PC Card Tuples
10H-1FH	identification Data	Identifies Card Type
20H-2FH	Identification Data	
30H-3FH	Identification Data	
40H-4FH	Compatibility Data (1)	Memory Technology #1
50H-5FH	Burst Data (1)	
60H-6FH	DRAM Data (1)	
70H-7FH	Reserved for future use	
80H-8FH	Compatibility Data (2)	Memory Technology #2
90H-9FH	Burst Data (2)	
AOH-AFH	DRAM Data (2)	
BOH-BFH	Reserved for future use	
COH-CFH	Compatibility Data (3)	Memory Technology #3
DOH-DFH	Reserved for future use	
EOH-EFH	Compatibility Data (4)	Memory Technology #4
FOH-FFH	Reserved for future use	

Table 4--Each MC *has an area of memory called the* **Attribute information Structure** *(AIS) that* **defines the type,** *amount, and operating* **characteristics of** *the memory on the card.*

feature means code can run in the card without being loaded into RAM.

SPLIT DECISION?

Today, CF is ahead in actual design-ins and production, but MC is marshalling a formidable response.

I like things about both cards. In particular, the smaller form factor (25% the size of PCMCIA) enables truly portable gear. Size limits discourage temptation to bloat the spec.

The controller fight seems excessive given the PC- and PCMCIA-centric presumptions that fuel it. For instance, how many desktop PCs have PCMCIA slots? There's plenty of PC software (OS, FTL, CFI, etc.), but what if you're making a card-playing gadget using a low-cost embedded micro?

Although list prices are high (-$50 per MB) and densities are just entering the double-digit range, no doubt the price will go down and density up. The cards may first appear in bleeding-edge products like digital cameras, but eventually they'll be everywhere.

Two standards may be best since it gives some choices and encourages competitive pricing. I'd rather deal with two simple, stable standards than one complicated, slippery one.

It doesn't look like a quick knockout, so your best bet is with the card you like. If your application is primarily flash-oriented and benefits from the transparency of a built-in disk-like interface, go with CF. If you expect to use nonflash memory technologies and need XIP, bet on an MC. ❑

Tom Cantrell has been working on chip, board, and systems design and marketing in Silicon Valley for more than ten years. Hc may be reached by E-mail at tom.cantrell@circellar.com, by telephone at (510) 657-0264, or by *fax at (510) 657-5441.*

SOURCES

CompactFlash Assn.
P.O. Box 51537
Palo Alto, CA 94303
(415) 843-1220
Fax: (4151 493-1871

CompactFlash
SanDisk Corp.
140 Caspian Ct.
Sunnyvale, CA 94089
(408) 542-0500
Fax: (408) 542-0503
http://www.sandisk.com/

Intel Literature Packet **#L154**
Intel Corp. Literature Center
P.O. Box 7641
Mt. Prospect, IL 60056
(916) 356-7368
Fax: (847) 296-3699
http://www.intel.com/embedded

Miniature Card
Intel Corp.
5000 W. Chandler Blvd.
Chandler, AZ 85226-3699
(602) 554-8080
Fax: (602) 554-7436
http://www.intel.com/

Miniature Card Implementers Forum
1900 Prarie City Rd., FM3-77
Folsom, CA 95630
(916) 356-7060
Fax: (916) 356-8607
http://www.mcif.org/
MinatureCardQccm.fm.intel.com

IRS

425 Very Useful
426 Moderately Useful
427 Not Useful

CONNECTIME

conducted by Ken Davidson

The Circuit Cellar BBS
300/1200/2400/9600/14.4k bps
24 hours/7 days a week
(860) 871-1988—Four incoming lines
Internet E-mail: sysop@circellar.com

It seems a week doesn't go by without my receiving a message from someone asking if they can get to the public message areas or the full file areas of the BBS via the Internet. Right now, the answer is no. However, we're working on a solution.

I'll divulge more details in the coming months, but in an effort to reduce some of my f-mail load, I wanted to let you know that such access really is on ifs way. Just be patient for a little while longer. Thanks.

The following message threads may be obtained in their entirety either from the BBS or on disk. See "Article Software" at the end of this column for more information.

Blinking neon lamps

I've wanted to use this thread for several months now, but have never had the room. It starts with a simple enough question: how do you make a circuit that flickers a neon lamp so it looks something like a candle burning?

That simple question touches off an extensive discussion of techniques, theory, and proposals that is almost amusing in its intensity at times. Definitely a fun diversion.

How to burn a GAL

For years, we at Circuit Cellar have received pleas from people to present a PAL programmer project. We've done EPROM programmers, so a PAL programmer can't be *that* hard, can it?

Well, when the basic device programming information just isn't available like it is for EPROMs, such a project takes on orders of magnitude more difficulty. Different chips require different programming algorithms, and manufacturers refuse to release detailed information.

This next thread starts with just such a discussion. You'd think that by bringing it up in a forum frequented by hundreds of engineers from around the world, someone might have some insight. Such wasn't the case.

Different people posit different theories as to why the manufacturers refuse to divulge details, and one overseas caller even posts a partial programming description in German, but nobody has the definitive answer. It makes for some interesting reading.

Metal detectors

In this age of AIDS and concern over blood- and fluid-borne diseases, hospitals can't be too careful in protecting its staff from accidental cuts and sticks. This thread starts with a question of how to scan a laundry bag for metal objects. The concern is that needles and other sharps objects that accidentally get into soiled linen bags at a hospital could injure workers handling the bags.

The only catch is the bags are loaded on metal carts with metal walls on three sides. Is there a way to scan for metal in the bags by simply passing the carts through some device?

Some interesting ideas are brought up, including the use of all-plastic carts, passing the bags through a metal detector after being unloaded but before being dumped out, and using the hospital's MRI equipment (it was a joke).

All the suggestions have one problem or another, and the issue isn't completely resolved. Do you have an idea? Take a look to see if it was already brought up.

Parallel Port Key

In the ultimate example of thread drift, this discussion starts with a question about parallel port dongles, and ends up talking about the use of different AC voltages and frequencies used in various countries and their affect on equipment.

How does such a transition take place? After some initial talk about the theory behind the need for dongles and how they work, someone suggests that they use a short cable to move the dongles away from the back of the PC to reduce the amount of clearance necessary behind the computer.

Another user points out that the cable length is limited by the port drivers used in the PC. He retorts with, "Okay, so you add some power FETs switching rectified and filtered 220 VDC from the mains. No problem with juice then, right?" It doesn't take long for someone to point out that 220 V isn't used in some countries and the entire discussion takes an abrupt turn to the new topic.

If you've wondered what voltages are used in other countries and why, this is a neat thread. Some of the input from our overseas callers is quite valuable.

CONNECTIME

Telephony design

Msg#: 3284
From: Jesus Lares To: All Users

I think I read somewhere in this BBS that if I wanted to design a telephony product (e.g., Caller ID box, small PBX, etc.) for commercial purposes (sell in small quantities), I could use DAAs with Part 68 registration like Xecom's or Cermetek's.

Can I slap together a controller, a DAA, and some software, and then sell the product without having to go through FCC certification or approval?

Msg#: 3288
From: Jeff Bachiochi To: Jesus Lares

Yup! You take on the DAA's certification. However, if your circuitry has any kind of oscillator over 100 kHz the system is considered a transmitter and must pass interference testing. They get you coming and going. Of course, the FCC is there to protect us from ourselves.

Msg#: 3290
From: Jesus Lares To: Jeff Bachiochi

Thanks for your reply. If I understand you correctly, I can use a 32.768-kHz crystal and a micro, and I'll be home free. Right?

Msg#: 3337
From: Jeff Bachiochi To: Jesus Lares

Last time I looked, that was the case. They don't seem to leave many loopholes like that. Good luck!

Msg#: 3307
From: Lyndon Walker To: Jeff Bachiochi

Shouldn't that be 9 kHz?

Msg#: 3338
From: Jeff Bachiochi To: Lyndon Walker

Do you know something I don't? We're not talking audio as in tones, but oscillator as in micro.

Msg#: 3370
From: Lyndon Walker To: Jeff Bachiochi

Quoting from 1996 *Compliance Reference Guide:* "The spectrum shall be investigated from the lowest radio-frequency signal generated in the device, not below 9 kHz, up to at least the 10th harmonic of the highest fundamental frequency or 40 GHz, whichever is lower."

This is from Part 15b certification section of Compliance Engineering's reference book. The actual section it's taken from is not mentioned, but the 9-kHz figure appears a number of times, so I don't think it's a typo.

Msg#: 3306
From: Lyndon Walker To: Jesus Lares

Yes and no. If you use a DAA with user-transferable registration, then you don't need *Part 68 * certification. However you may very well need Part 15 certification which is a whole other ball of wax.

Msg#: 3312
From: Jesus Lares To: Lyndon Walker

Thanks for your reply. I do not know if you are familiar with FCC certification procedures, but I will ask you the following question anyway. Is Part 15 certification within a small budget range (hundreds) or is it out of the question (thousands)? I ask you this because, I used to work for a company that developed a call controller, and they paid around $10,000 for certification (both Part 68 and 15).

Msg#: 3369
From: Lyndon Walker To: Jesus Lares

Most of the money will be for the testing lab you send your device to. Part 15 certification fees are less than $1000 and Part 68 is $180. These are fees paid to the FCC.

As far as the cost of having the device tested, I've never done it myself. But from memory of when we looked into it a few years ago, the price was around $5000 assuming all went well the first time around.

Msg#: 3355
From: George Novacek To: Jesus Lares

It depends. The certification itself is not that expensive. It will be in the hundreds. But you must submit a test report with your application. And that is when it is going to cost you in thousands to tens of thousands, unless you have the equipment to do it yourself.

You also must be qualified. At one point, just a registered professional engineer was good enough. At some point, I am given to understand (I have not dealt with FCC for 7 years), the engineer needed to get on the list approved by FCC.

Last, if you go to an independent lab and things go wrong, watch out! At roughly $1000/day it is very easy to rack up a huge bill for retesting or in-lab development.

CE approval

Msg#:11433
From: Brian Farmer To: All Users

We are a manufacturer of light industrial equipment used in Europe. We would like to gather information regarding the most cost-effective way to obtain CE approval. At this point, we believe there will need to be some modifications to our equipment. Anyone with experience in this area, please respond. Thank you.

Msg#:14525

From: Pellervo Kaskinen To: Brian Farmer

Based on our experiences, there may not be a truly ″cost-effective" way.. .. The problem is the general unavailability in this country of the basic standards. I mean.. .they are available, but not in the public libraries and similar free-of-charge form. I think we spent about $7,000 for our set. We started with just a couple of the documents, by recommendation of our European office. That gave references to further documents that we believed would be relevant.

We tried to be very selective, but after getting the next set, we found further references that required additional purchases. Luckily, after the third round we were pretty well stocked with what we needed.

There is a certain amount of information in *Compliance Engineering* Magazine. It is free for qualified persons. In fact, it contains ads for the testing labs, the instrument makers, and for the bodies capable of providing you with the European standards. Looks like one of the more aggressive document outlets nowadays is the Danish Testing Authority (whatever its actual name.. .). They sell the EN*xxxxx* documents in English versions, despite their nationality.

If you are using UL for anything, your best bet might still be using them-they are building up their EU connections.

I'm sorry, but I cannot give a comprehensive answer, because the *relevant* rules vary by the type and by the intended market of your product. I would say, though, that you have to meet the Low-Voltage Directive in any case.

Noise rules vary, but most small appliances and similar products already have to meet the power factor and harmonic content rulings in addition to conducted and radiated noise limits. The scope of the harmonics and power factor rulings is going to be expanded in June 1998 to cover virtually all items using AC power. Presently, below some 80 W and above 16 A (at 230 V) are outside of the scope.

Besides the limits on emitted noise, there are requirements for noise *immunity.* The rules on that front generally do not require complete absence of any malfunctioning, but instead that the failures do not cause danger. Your product must tolerate certain power-line transients and transients coupled to any and all data lines and similar that are longer than about 6 ft. The length requirement is implicit because the noise is injected to the cable by a I-meter trough, through which your cable is run. Cables that are too short for the coupling device are excused.

Being excused does not mean they do not have to meet some noise-immunity tests. The static-discharge test is to be applied to the peripheral devices as well as to the main console (in our case).

A big hassle is the labeling and the instruction area. The basic idea is that you either have to provide native language warnings in whatever country your product is sold, or use graphical symbols and explain them in the manual at the native language.

First, the symbols have to be chosen from a set of predefined (IEC or ISO) sets or they have to be designed to meet rather stringent guidelines regarding their absolute and relative sizes and other details. I think some of the documents you should consult are IEC 447 and IS0 7000.

Oh, one last thing you absolutely have to cover is the test-voltage levels. It is a bit complicated because you have to define the use types and “pollution” categories (water and dust effects). And this leads to direct and along-the-surface distance requirements of all line-side traces that generally exceed what the American designers have grown accustomed to.

The transformer primary-to-secondary insulation test here is typically 1,500 V, but a rough estimate is that the European rules are going to be satisfied only with a 3750-VAC, l-minute test. Or even higher, if the pollution level appears to require tightening of the rules.

A related requirement is that the yellow/green protective ground lead be able to carry enough current (25-A test min.). And it has to be terminated in a properly marked (PE = Protective Earth) stud that is not used for any other purpose.

I hope this gives you a starting point. If you have more specific questions, I’ll try to help, but be aware that the “devil is in the details.” What looks like a completely plausible set of rules can be totally invalidated by a minor overlooked condition of intended use and so on.

Current limiters

Msg#: 7139
From: Mark Delaune To: All Users

I am looking for a foldback current limiter. I was thinking about using a differential op-amp sensing current through a resistor and taking the output of this into a comparator with lots of hysteresis to shut the supply down.. .or perhaps to fire an SCR once the comparator switches, making the user recycle power. Any ideas!

Msg#:11693
From: Pellervo Kaskinen To: Mark Delaune

What kind of power supply are you talking about? The foldback limiting is common on linear power supplies, while switchers typically shut completely off. After a few milliseconds they try a new (possibly soft) start.

Firing an SCR to short circuit the output of a power supply is a last resort as far as I’m concerned. You tend to get more than you bargained for. The SCR does not short the output completely; there is likely to be a volt or two remaining. If the power supply has a 5-V, 15-A capacity, your SCR has to dissipate 20– 30 W. A big heat sink.

In fact, the SCR is usually paired with a fuse and only intended for cases where the pass transistor is shorted and would let the whole supply voltage reach the 5-V circuit. Then, it is better to short the supply, blow the fuse, and protect the rest of the circuits.

If you suspect a load-side short circuit or overload, then the current limiting with foldback characteristics is an attractive choice. Minimize the overall power dissipation, especially on the pass transistor.

The one most important side effect from the foldback characteristics is that often the loads require quite a bit of inrush current when turning on. A foldback limiting by its very nature provides a reduced current at this situation and may prevent the load from starting altogether. I had such an experience a long time ago and have since favored straight current limiting without the foldback.

Sources for optical interface circuits

Msg#: 3878
From: Mike Aulik To: All Users

I’m hopeful someone can point me in a direction.

I am on a contract project for a medical school. I need names of companies that make ICs and/or interface hardware that will allow a beam of light over a fiber-optic cable to act as a switch. Sources for the cable and any related components would be greatly appreciated. Also, if anyone can suggest sources for design information in this area, I would greatly appreciate that as well. Thanks much.

Msg#: 3976
From: Pellervo Kaskinen To: Mike Aulik

Simple answer: Microswitch division of Honeywell.

I believe there are several more, like Banner, but truly the first name that comes to my mind is Microswitch.

Moreover, they are quite well represented all over the country. You can go to an electrical contractor supply house and it's likely that they will have at least the necessary catalogs. Or you can go to any one of the myriad electronics distributors. They'll have Microswitch among their sources.

Msg#: 3882
From: Lyndon Walker To: Mike Aulik

What kind of switch? What distance are you running? How fast do you need to switch!

For relatively slow (to a few 100 kHz) optical communication over short distances, 1000-µm plastic fiber works well and is easy to use. Look at the AMP Optimate DNP connectors. They also make plastic fiber. I think Digi-Key still carries them.

Siemens makes optical connectors with integrated detectors and emitters. Motorola used to have a similar system, but they just made it obsoleted.

Plastic fiber is especially attractive because it doesn't require special tools and is very inexpensive. I was paying about 75¢ per meter in small quantities three years ago.

If you just need, say, a low-side power switch controlled from fiber-optics, two transistors, a few resistors, and the Motorola FLCS (can still be purchased if you just need one or two systems] optics are all you need. Cost less than $10 total. For ultra cheap, just drill holes into LEDs and photodetectors to make your own optical connectors. AMP Optimate takes a bit more work, but it's still pretty easy to use.

For the more sophisticated, faster "real fiber" stuff, look at HP, Siemens, and Honeywell.

As far as design info, the Motorola Optoelectronics books from '93 had sample circuits. A distributor may still have a few of these older books. AMP has a good intro, *A Technician's Guide to Fiber Optics.* Digi-Key carries it (or used to). It's a basic intro to fiber, not a design manual.

We invite you to call the Circuit Cellar BBS and exchange messages and files with other Circuit Cellar readers. It is available 24 hours a day and may be reached at (860) 871-1988. Set your modem for 8 data bits, 1 stop bit, no parity, and 300, 1200, 2400, 9600, or 14.4k bps.

ARTICLE SOFTWARE

Software for the articles in this and past issues of *Circuit Cellar INK* may be downloaded from the Circuit Cellar BBS free of charge. It is also available on the Internet at **http://www.circellar.com/**. For those with just E-mail access, send a message to **info@circellar.com** to find out how to request files through E-mail.

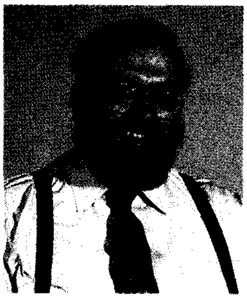
Message threads summarized at the beginning of the column are also available on the BBS for at least six months after they are first posted. The subject line at the start of each summary matches the subject used on the messages themselves. Simply call the BBS and search for those subject lines to find the message threads.

For those unable to download files or messages, the software and messages are also available on disk. Software for issues prior to 1995 comes on a 360-KB IBM PC-format disk, one issue per disk. For issues from 1995 on, software comes on a 1.44-MB PC-format disk, with three issues per disk. Disks cost just $12 each. To order Software on Disk, send check or money order to: Circuit Cellar INK, Software On Disk, P.O. Box 772, Vernon, CT 06066, or use your Visa or Mastercard and call (860) 8752199. Be sure to specify the issue numbers with your order. Please add $3 for shipping outside the U.S.

428 Very Useful 429 Moderately Useful 430 Not Useful

PRIORITY INTERRUPT

Bits and Bytes



We all have aversions to certain things in life. As I get old and crotchety, I worry less about apologizing for what I don't like and considerably more inclined to seek out exactly what I do like. If Bob Pease (see last month's editorial) can count dead cars, I guess I can complain about how much I hate grass maintenance and airline travel.

What do they have to do with computer applications? Nothing, I suppose. Going to the Embedded Systems Conference in San Jose reminded me how much I hate crowded airports. And, while I was on the subject, I just thought I'd throw in the fact that lawns aren't my thing either.

Realize, of course, I'm just speaking for myself. I love driving through Litchfield, CT where all the homes have baseball-park-sized, scissor-manicured lawns. I'm just glad all that grass is theirs, not mine. However, anyone analyzing the effort that I go through in maintaining a nongrass groundcover would laugh themselves silly. I'll save that story for another time.

While I perpetuate the nonexistence of my grass, it's invariably my publisher, Dan Rodrigues, who makes commitments that cause me to end up on planes heading west. Needless to say, this leads to getting even.

There's nothing more fun than tormenting a publisher. When I returned from the conference, I saw the perfect opportunity. Half hunching over Dan's desk, I remarked with a pained expression, "Dan, I've got some good news and some bad news about the Embedded Systems Conference!"

He looked up from his spreadsheet. From experience, he knew that hearing bad news first wasn't going to be music to his accountant ears. Out of fiscal necessity, however, perhaps he'd better deal with it. He responded soulfully, "OK, what's the bad news?"

Conjuring up my most disgusted and exasperated expression, I said in a defeated tone, "We only sold three magazine subscriptions at the show!"

Just like the old joke about the engineer who offers to fix a broken guillotine (guess who's next), you can always count on a publisher with an accounting background to instantly react to such news by grabbing his chest and gasping for air. Twelve grand in expenses divided by three subscriptions! Arrrghh!

But, Dan's getting smart enough by now that he knows when I'm trying to scramble his abacus. He banged his hand on the desk and stated emphatically, "Wait a minute! You weren't supposed to be selling any subscriptions! Everybody in that place is already an *INK* reader!"

"Almost gotcha," I smiled. "Financial guys are so easy."

As he was muttering something about "dumb engineers," I couldn't help but beam at the reality of *INK*'s success and our experience at the show. The only reason for the three subscriptions was that they came from foreign travellers who were worried that they might miss an issue if the subscriptions weren't placed eye-to-eye. More than half the exhibitors were advertisers, and virtually all the attendees were readers. There wasn't a booth I entered where they didn't know *INK* and welcome us.

Although I might protest for weeks before getting on a plane to a convention, these days without exception I come back knowing that what I've been doing all these years personally and with *INK* hasn't gone unnoticed. More and more people have made me aware that the course we're on and the knowledge we consistently present has positively affected their professional lives.

The only slightly pessimistic comment came from an industry maven who suggested that my decidedly conservative nature may in fact be holding *INK* back from realizing its real (i.e., financial) potential. I reminded him that, in my experience, magazines that had responded solely to stockholder demands are lucky if they're a mere bit of their former BYTE. Call me conservative if you must, but we still have our hands on all eight.

Steve

steve.ciarcia@circellar.com