Advanced Micro Devices

1985

Annual Proceedings

A Compendium of Technical Articles
and Conference Papers

Advanced Micro Devices

1985

Annual Proceedings

901 Thompson Place, P.O. Box 3453, Sunnyvale, California 94088
(408) 732-2400 TWX: 910-339-9280 TELEX: 34-6306

Printed in U.S.A.

ISSN 0883-1262

CREDITS

"Denser process gets the most out of bipolar VLSI,"
"Videotex hardware heralds another node—the home,"
"Third-generation codecs pave way for future digital networks,"
"Data-compression IC saves on storage and speeds up image transmission,"
"Protocols and network-control chips: a symbiotic relationship,"
"Protocol standardization works its way up the ladder of the OSI model,"
"New standards for local networks push upper limits for lightwave data,"
copyright 1984, McGraw-Hill Inc. All rights reserved. Reprinted by permission.

"A high performance CMOS process for the next generation EPROM,"
"Design of a Single-Chip Bell 212A Compatible Modem Employing Digital Signal Processing,"
copyright 1984 IEEE. All rights reserved. Reprinted by permission.

"Surface Mount Technology, Plastic Chip Carrier Trends" copyright 1984, Lake Publishing Corp. All rights reserved. Reprinted by permission.

"La conception d'une unité centrale microprogrammable" reprinted by permission of *Electronique, Techniques et Industries.*

"A new concept in backplane communication,"
"Super-minicomputer design using PALs*" reprinted by permission of *Electronic Engineering.*

"Putting the speed into 12-bit A-to-D conversion" reprinted by permission of *New Electronics.*

"Microprogrammable chips blend top performance with 32-bit structures,"
"Bipolar building blocks deliver supermini speed to microcoded systems,"
"Single-chip accelerators speed floating-point and binary computations,"
"One-chip sequencer shapes up addressing for large FFTs,"
"Controller chips pair up in S interface for ISDN systems,"
"Monolithic color palette fills in the picture for high-speed graphics,"
"Three chips, plus palette, aid graphics designers to paint detailed pictures,"
"64-kbit EEPROM speeds in-system reprogramming, adds data polling,"
"Logic-programming language enriches design processes,"
"RAM's on-chip registers build simple control stores that include self-diagnostics," reprinted with permission from *Electronic Design*; copyright Hayden Publishing Co., Inc. 1984.

"High-performance, 32-bit building blocks address a broad spectrum of applications" reprinted by permission of *Nikkei Electronics.*

"Digitale Signalverarbeitung mit bipolaren LSI-Bausteinen" reprinted by permission of *Elektronik Industrie.*

"500-kHz single-board FFT system incorporates DSP-optimized chips,"
"Design real-time Ethernet systems by giving priority to nodes,"
"Use a CRT-controller chip to mix text and graphics," reprinted with permission from *EDN*; copyright 1984 Cahners Publishing Co. All rights reserved.

"Very-high-speed floating-point processor,"
"Data security with fast, high-integration circuits" reprinted by permission of Electronica.

"Data encryption: high-speed implementation of DES" reprinted by permission of *Communications Engineering International.*

"Standardizing upper-level network protocols,"
"Two-chip set tackles disk control problems,"
"Winchester/floppy controller eases disk interfacing," reprinted with permission from *Computer Design.* Copyright Pennwell Publishing Co., 1984. All rights reserved.

"VLSI and digital networking,"
"DRAM controller helps scrub errors,"
"Resettable RAMs speed cache systems," reprinted with permission. Copyright Hearst Business Communications, 1984. All rights reserved.

"Nuove architetture per accelerare la gestione dei pixel nelle applicazioni grafiche" reprinted by permission of *Elettronica Oggi.*

"The Role of Programmable Logic in System Design" reprinted with permission from *VLSI Design.* Copyright CMP Publications 1984. All rights reserved.

"Logik-Baustein kombiniert PAL- und Gate-Array-Technik," reprinted by permission of *Elektronik.*

"Methode de test et de conception de diagnostic," (parts 1 and 2) reprinted by permission of *Minis et Micros.*

*PAL is a registered trademark of and is used under license from Monolithic Memories, Inc.

ADVANCED MICRO DEVICES 1985 ANNUAL PROCEEDINGS

A Compendium of Technical Articles and Conference Papers

CONTENTS

CONTENTS

CONTENTS

PREFACE

The articles and conference papers included in this volume are the best examples of the products, technology, and applications that have made Advanced Micro Devices a leader in the integrated circuit industry. These articles and conference papers from 1984 are organized by the major application areas that incorporate AMD chips, such as telecommunications, networking, graphics, digital signal processing, and mass storage. We are publishing them to underscore AMD's substantial contributions to computation, instrumentation, and telecommunications technologies.

Founded in 1969, Advanced Micro Devices is a leading worldwide supplier of proprietary and industry-standard semiconductors for manufacturers of electronic systems. The company produces a broad range of devices, including microprocessors, peripherals, memories, telecommunications and logic products, and semicustom integrated circuits. With state-of-the-art bipolar and CMOS manufacturing capabilities, AMD is in a unique position to draw on the strengths of both technologies to provide the products demanded by an increasingly sophisticated market. At the same time, the company offers the highest quality guarantee in the industry for every one of its products.

AMD is the fastest growing U.S. semiconductor manufacturer, with a 10-year compound annual growth rate of 42 percent. Fiscal 1985 ended with net sales of over $930 million, up from $583 million for fiscal 1984. AMD employs over 15,000 people worldwide, and has manufacturing facilities in Sunnyvale and Santa Clara, California; Austin and San Antonio, Texas; Woking, England; Penang, Malaysia; Manila, Philippines; Bangkok, Thailand; and Singapore. AMD's corporate headquarters are in Sunnyvale.

Denser process gets the most out of bipolar VLSI

To shrink bipolar chips to feature sizes of very large-scale integration, IMOX-S technology implements a series of closely related processing steps

by Phil Downing, Pete Gwozdz, and Bernie New, *Advanced Micro Devices Inc., Sunnyvale, Calif.*

□ Design considerations for new integrated systems range widely, from projecting on-board power consumption to estimating production costs, but the principal benchmarks of performance are the speed and density of individual large-scale integrated circuits. Though significant improvements in system size and capability are nothing new, it is only with the most recent bipolar processing techniques, such as IMOX-S (for ion implantation, oxide isolation with scaling), that the benefits of bipolar speed have come to very large-scale integration (see "Designing high-performance systems with newer, faster processors," below).

Shrinking size, power use

High-speed IMOX technology, introduced in 1980, has been incorporated into such products as the 2901 bit-slice microprocessor and the 29116 bipolar microprocessor, as well as into various programmable read-only memories and logic chips. When development began on IMOX-S, two principal goals were an improvement in speed and in gate densities: chips produced with it were to be denser, incorporate smaller geometries, and be significantly faster than earlier IMOX parts or other bipolar ICs (see table). At the same time, the designers of the new scaled process wanted to avoid the increase in power level that usually accompanies higher processing speeds, and this they brought off as well.

At first, it may seem impossible to satisfy these usually incompatible requirements. IMOX-S, however, succeeds by concentrating not only on the primary contributions of each processing step but also on how the successive steps interrelate. These interrelationships—one of the more interesting aspects of the IMOX-S process—work synergistically so that the final process is more than the sum of the individual steps.

Like many other processes, IMOX makes use of oxide isolation for device sizes that are smaller than older, junction-isolation processes could achieve. But the designers of other processes opted for an unwalled-emitter structure that requires extra silicon area (the absence of which would allow the emitter and collector regions to make electrical contact, thus incapacitating the transistor) to isolate the collector and emitter regions.

IMOX instead uses oxide isolation to wall off the transistor's emitter (Fig. 1), eliminating the wasted area found in the unwalled-emitter approach—the field oxide defines three of the emitter's four edges. The direct contribution of oxide isolation to shrinking device size is considerable; when combined with a walled-emitter structure, the result is dramatic.

Speed, too

Oxide isolation and walled-emitter structures have the advantage of speed, as well as of greater density. Device speeds are faster because some of the capacitances that usually reduce the switching speed have been eliminated through the reduced device area. The collector-to-base capacitance usually accounts for 50% of the total delay

Designing high-performance systems with newer, faster processors

Architectural enhancements such as microprogramming in very large-scale integrated circuits can allow one processor to do more useful work than another in a single instruction cycle—but, as a rule of thumb, the faster a processor executes instructions the better. Hence the single most telling specification for any processor, and potentially the most demanding performance requirement, is the instruction-cycle time.

For pure speed, the 29116A 16-bit microprogrammable processor and the 29517A pipelined 16-by-16-bit multiplier—both fabricated with the IMOX-S process—should be considered for the core of high-performance dedicated systems such as graphics processors. The 29116A and the 29517A can execute instructions in as little as 80 nanoseconds—more than 10 times as fast as a fixed-instruction-set microprocessor for similar instruction complexity.

Besides the more familiar logical and arithmetic instructions, the 29116A has the type of instructions necessary for image-oriented graphics algorithms. For example, single-clock-cycle instructions can rotate from 1 to 16 places and merge two 16-bit data fields on a bit-by-bit basis, as well as prioritize a 16-bit data value. In addition, the 29517A, through inclusion of pipeline processing, executes a 16-by-16-bit multiplication with a 32-bit result in one clock cycle.

Process	Nonoxide-isolated processes	IMOX	IMOX-S
Technology	Contact alignment Isotropic etch Two-layer metal	1:1 projection alignment Isotropic etch Two-layer metal	Stepper technology Anisotropic etch (including metal) Three-layer metal
Minimum drawn feature size (μm)	4.0	3.0	1.5
Metal pitch (μm)	11.0	8.0	4.0
In-circuit gate delay at 1 mW (ns)	—	1.5	0.8
Transistor maximum operating frequency (f_T) (GHz)	—	2	2

when operating at 1-milliwatt gate power, but, because the walled emitter directly reduces the base size, the associated collector-base capacitance drops.

The use of the walled-emitter transistor structure neatly reduces device size; complementing this feature of the process architecture are fabrication techniques—most prominently the use of stepper lithography and anisotropic etching. Compared with more traditional masking techniques, steppers cause minimal variations over a wafer's surface by exposing wafers a few die at a time. In addition, misalignment problems are less severe because a few die at most are aligned at a time, whereas traditional methods align the entire wafer at once.

Reactive ion etching

Anisotropic etching (Fig. 2) controls the direction o the removal of material from the wafer surface; the usua means is either a plasma or reactive ion-etching process, which is the process used for IMOX-S. Side etching is minimized with anisotropic etching, which removes only the material directly below the mask opening.

Since anisotropic etching allows the smallest feature size while wafer-stepping technology draws fine line widths, the overall result is a much smaller device—one that can be defined to the full limit of the lithographic process, not only because of the individual benefits of each step but also because of the synergistic interrelationship of the steps.

The IMOX-S transistor structure, fabricated with a combination of the above techniques, has a number of advantages over transistors fabricated with the older diffusion-based processes. The ion-implanted base and emitter regions intersect at a much lower concentration level than the diffused-base structure. Because dopant levels are lower, the base-emitter capacitance per unit is lower, reducing delay. Furthermore, the transistor's gain, or beta—which primarily depends on the total amount of dopant deposited in the base region—is much more controllable in the ion-implanted structure.

In a transistor fabricated with a diffusion-based process, which has a high level of dopant concentration at the intersection of the base and emitter, a shift in the intersection causes a dramatic change in the base dopant. The implanted structure, with its lower intersection concentration, will be less sensitive to these shifts, will have tighter beta tolerances, and hence will be faster.

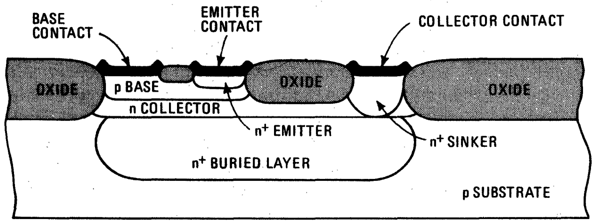
One of the differences between the IMOX-S transistor and other modern transistor structures is the use of a low-resistance plug, or sinker, directly below the collector contact. In IMOX-S, the usual buried layer of low-resistance silicon alone does not reduce the collector resistance to levels consistent with the process's speed and power requirements. Concern for collector resistance grows as transistor size shrinks, but the addition of the implanted sinker provides a low-resistance path from the contact to the base-collector junction.

Hooking up

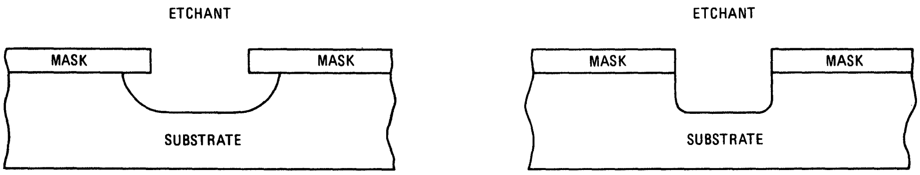
Once device sizes are at the minimum, there is little extra space over which to route the necessary interconnections. In addition, each metal line placed on the surface occupies precious silicon area, which could have been used to implement transistors. In IMOX-S, three-layer-metal technology places a large portion of the interconnection off the silicon surface, making interconnection easier and less expensive since it wastes less die area.

Before, multiple-layer-metal techniques required wide, thick metal lines to compensate for the discontinuities, or steps, in the silicon surface left by previous etching and diffusion steps. The die surface lost the planarity, or smoothness, necessary for thin, narrow metal lines. As Fig. 3a shows, steep steps in the wafer surface might cause metal lines to narrow and tear. Thicker metal lines would be necessary to ensure reliable metal interconnection. Thicker lines, however, tend to make the surface even more ragged, which makes top layers of metalization even more difficult to apply.

But the IMOX-S process uses carefully selected oxide growth and etch steps to retain the surface's planarity



1. Walled in. Rather than using up precious silicon real estate to electrically isolate the transistor's collector and emitter regions, IMOX-S surrounds the emitter with a wall of oxide. The sinker, or low-resistance plug, facilitates current flow to the buried layer.



2. Perpendicular etching. Isotropic etching (a) removes material in all directions below a mask, making the feature size larger than the limit of the lithographic process. But the anisotropic-etching technique (b) used in IMOX-S burns straight down into the wafer.

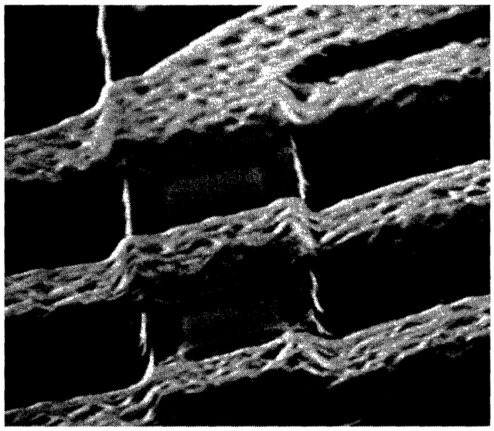
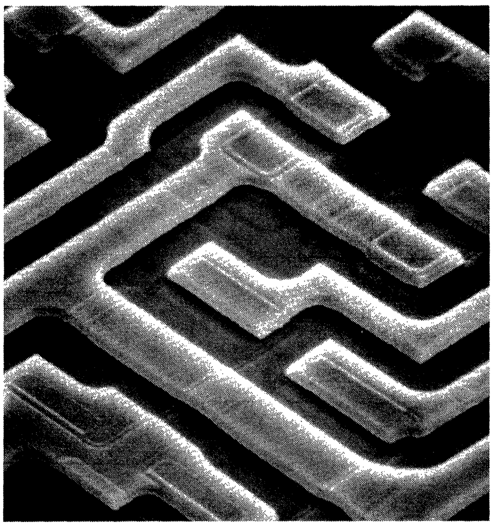
(Fig. 3b). The first metal layer can be placed on this planar surface much more reliably and with tighter tolerances, thereby contributing to the overall increase in density.

Smaller metal lines also contribute indirectly to improved performance: if the lines are narrower, because of improved lithography and processing techniques, and shorter, because of higher device density, then the capacitance associated with each line is reduced. Lower capacitance makes higher performance levels possible, especially on high fan-out transistors such as internal output drivers on bus-organized VLSI chips.

Second-layer metal now contends with the newly discontinuous surface introduced by the addition of the first metal layer. A planarization technique called one-to-one etching is used to remove this new discontinuity. The technique first creates an oxide layer over the entire wafer surface, covering metal as well as silicon. A photoresist—which is specially etched at the same rate as the new oxide covering—is spun on with a centrifuge in order to create a smooth surface, and the surface is then etched until only oxide remains.

Because the photoresist etches at the same rate as the oxide, the resulting oxide surface has the desired smooth planar quality. Hence the second layer of metal has the thin and narrow qualities necessary for a high-density VLSI process.

The third layer of metal in IMOX-S chips is used almost exclusively for power and ground buses—hence a thick, wide metal line is desirable to reduce the resistance of the metal and thus minimize unwanted voltage drops along these critical lines. A planar surface is no longer necessary after third-layer metal since no critical deposition steps follow.



3. Smooth. The first layer of metal interconnection for a conventional bipolar process (a) is much more ragged than in IMOX-S (b). The smoother the silicon and oxide surface below the metal layer, the more reliable the finished integrated circuit will be.

Owing to the huge commitment of capital and labor for new process development, it is important that an evolutionary path be followed so that it is easy to add future improvements in lithographic and diffusion technologies. IMOX-S can easily use new techniques to further reduce feature sizes because all the critical parameters of its transistor structure can be shrunk or scaled. In fact, the transistor structure was designed with this characteristic in mind.

Further improvements will come with the IMOX X-85 process, with its slot-isolation technique—a narrow groove cut into the surface of the silicon wafer and filled with an insulator to isolate the adjacent active elements. IMOX X-85, now under development, will have a minimum feature size of 1 micrometer and a 2.5-μm metal pitch, meaning that device size can be cut approximately in half. Yet these new devices to be fabricated with IMOX X-85 will deliver twice the performance of those fabricated with the IMOX-S process. □

Bipolar and CMOS Technologies Become Partners In Digital Signal Processing

by Robert Perlman
senior engineer
Product Planning and Applications
Advanced Micro Devices
Sunnyvale, CA 94088

Abstract

Bipolar and CMOS technologies each offer distinct advantages to those interested in implementing high-performance digital signal processing systems. For applications requiring high operating speeds, bipolar remains the process of choice; for applications requiring low power and more modest speeds, CMOS is suitable. By making judicious choices, the digital designer can use both technologies to produce an optimal design.

Introduction

Until a few years ago, designers of high-performance digital signal processing systems had little choice but to use bipolar technology. While the CMOS devices available at that time had the low power that many system architects desired, those devices were far too slow for serious consideration.

Today, although bipolar technology is still the solution for systems demanding the very highest performance, the advent of fast silicon-gate CMOS parts gives the designer a reasonable alternative when designing a system with somewhat more modest speed requirements. Bipolar and CMOS technologies differ in a number of significant ways. These differences have architectural implications that the designer must be aware of when choosing a technology or technologies for implementation of a digital signal or array processing system.

Comparing the technologies

Speed - While it's generally agreed that bipolar technology is faster than CMOS, there's some disagreement as to just how much faster it is. Estimating the speed advantage of bipolar over CMOS is often a difficult task for the casual observer because, invariably, one has no choice but to compare an emerging technology with a mature one, or to compare devices with different functionality or architecture.

A case in point is a comparison between the AMD Am29517A 16 x 16 bipolar multiplier and the soon-to-be-introduced Am29C517 CMOS multiplier. The bipolar part has a clocked multiplication time of 38 ns over the commercial temperature range, the CMOS multiplier a time of 65 ns. One might rashly assume that the bipolar device is only seventy percent faster. This assumption, however, ignores the fact that the internal architectures are different. The bipolar part does not use a recoding algorithm, and forms the product by summing sixteen partial products, while the CMOS part uses two-bit Booth recoding and fast adder techniques to improve speed. A bipolar device designed with two-bit Booth and fast adders, and implemented with AMD's IMOX-S (for ion implantation, oxide isolation with scaling) process would be significantly faster than the Am29517A.

The actual speed difference between the fastest bipolar and the fastest CMOS parts is a factor of two or three. While this figure is only a rule of thumb, it finds its origins in AMD's experience with both technologies.

Power - A device's power comprises two terms - static power, which is dissipated when the device is quiescent, and A.C power, which is dissipated when the device's nodes toggle.

For applications demanding low static power, CMOS devices are clearly superior . The static power of a CMOS device is usually negligible; this low static power, appealing as it may seem, is rarely a consideration in high speed digital signal processing systems, as the devices in such systems are usually clocked at as fast a rate as is feasible. Much more significant is the contribution of the A.C. power term, which is roughly proportional to the system clock frequency. If the clock frequency is high enough, CMOS devices may actually dissipate more power than their bipolar counterparts - when comparing low-power Schottky with HC CMOS, for example, the cross-over point is in the neighborhood of 10 MHz. In many cases, it may be difficult to estimate the A.C power of a given CMOS-based system by means other than direct measurement, because the power is dependent on the average percentage of nodes that toggle during a clock cycle, this percentage varying from system to system.

Bipolar devices typically consume from one to ten times the power of CMOS devices, the exact ratio depending on the technologies being compared and the operating speed. Bipolar devices using internal ECL (which include most of AMD's newer bipolar VLSI) draw more or less constant power, although capacitive effects and output driver totem pole currents give rise to a small speed-dependent power term.

Functional density - The functional densities of CMOS and ECL-internal bipolar devices are roughly the same. Although MOS is usually thought of as a high-density process, there is considerable difference in the densities of the various MOS technologies. NMOS, for example, has a density advantage over bipolar, but CMOS gates contain more devices than their NMOS counterparts. Bipolar design innovations, such as walled-emitter structures, allow bipolar densities to keep pace with CMOS.

One factor working against CMOS is that architectural and logic design concessions must often be made to improve speed; these compromises in turn reduce functional density. A case in point is the problem of CMOS gate width - CMOS gates tend to become slower as more inputs are added. A designer, then, might be tempted to replace a logic implementation that uses multiple-input NOR gates with one having only two-input NOR gates, thus improving performance but increasing the number of gates and the die size.

Although CMOS functional density is no greater than that of bipolar, it may be possible, in some cases, to make a larger die and pack a greater number of logic functions using a CMOS process. The reason for this is lower power consumption: it's much easier to meet package power limits with CMOS than with bipolar.

Drive - Bipolar gates have somewhat better drive characteristics than CMOS gates, particularly when the line being driven is capacitive. This superiority holds both for drive within a chip and for drive between chips.

Transmission line effects - Because new CMOS parts have edge rates approaching those of bipolar parts, CMOS users are now encountering transmission line effects similar to the ones that have plagued TTL designers for years. The problem is somewhat less severe than that for the highest speed TTL families, due to the large voltage noise immunity of CMOS, and to the ESD diodes found in many CMOS parts, which tend to clamp reflections to either supply rail. Even so, the problem is serious enough to deserve the designer's attention.

At present, only bipolar technology offers a clean, predictable means of controlling transmission line effects in the highest-performance systems - this means is ECL I/O. The high-drive, low-impedance emitter follower outputs and the high-impedance differential amplifier inputs of ECL allow the user to series- or parallel-terminate signals correctly, thus reducing reflections.

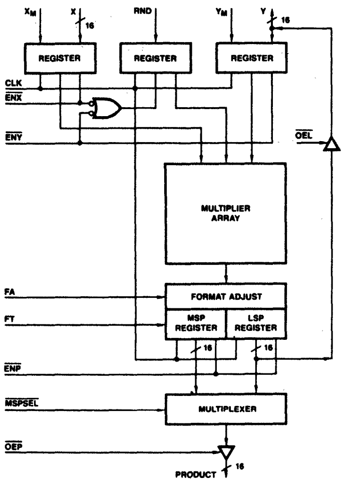
Noise immunity - Comparing the noise immunities of bipolar and CMOS yields no clear winner. At first glance it may appear that CMOS prevails, as its voltage noise immunity is roughly Vcc/2, while bipolar noise immunity is usually on the order of hundreds of millivolts. The conclusion one might draw from this comparison is deceptive for two reasons. First, energy noise immunity is more crucial an issue than voltage noise immunity. Because CMOS gate inputs have a much higher impedance than their bipolar counterparts, it takes much less noise energy to drive a CMOS gate into false transition. Second, CMOS logic thresholds are set by both the supply voltage and ground; bipolar logic thresholds depend mostly on ground, making bipolar logic somewhat less susceptible to power supply noise. This is not to say that bipolar logic is more noise-immune than CMOS, but to point out that the often-alleged superiority of CMOS in this area is by no means clear-cut.

Architectural implications for integrated circuits

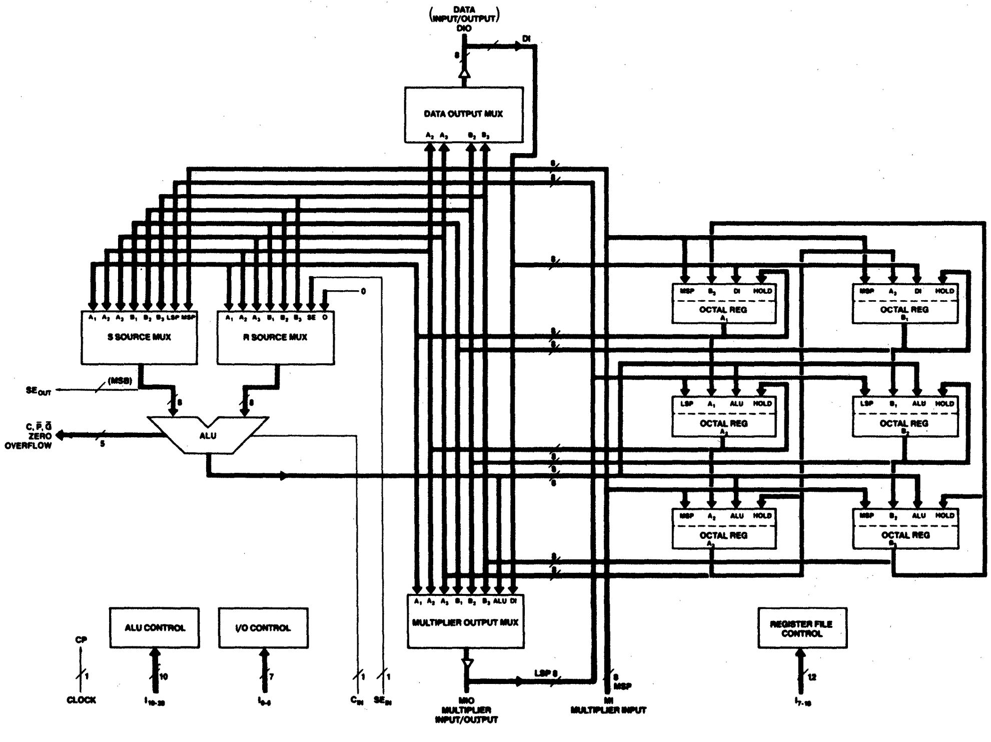
The differences between bipolar and CMOS technologies have two major implications for the architecture of integrated circuits made with these processes:

Local vs. global communication - Integrated circuit architectures can be divided into two broad categories. Into the first category fall those chips in which communication takes place primarily between neighboring circuit elements; these architectures feature local communication. An example of an integrated circuit that depends heavily on local communication is the Am29517A 16 x 16 combinatorial multiplier (fig. 1a). Most communication in this device takes place among neighboring computational elements in the multiplier's combinatorial core.

The second category includes those architectures in which widely-separated circuit elements talk to one another over buses; such devices feature global communication. The Am29501 Multi-Port Processor (fig. 1b) is an example of a global communication architecture, with its many resources - multiplexers, registers, and arithmetic unit - communicating over a large number of internal buses.



(a.)



(b.)

Fig. 1: Architectures using local and global communication.
(a.) the Am29517A 16 x 16 multiplier -local communication;
(b.) the Am29501A Multi-port Processor - global communication.

Due to the limited drive capability of its gates, CMOS is most suitable for integrated circuits that depend heavily on local communication. Bipolar excels in designs that need a significant amount of global communication at high speeds.

Pipelining - Because CMOS gates have speeds and drive capabilities that are somewhat more modest than bipolar, designers of CMOS integrated circuits often insert pipeline registers at strategic points in data and control paths to maintain a reasonable throughput rate.

Pipelining is a time-honored technique for improving system speed, and is not necessarily a disadvantage. Excessive pipelining, however, particularly in the middle of arithmetic elements, can create difficulties when programming signal processing algorithms for which intermediate results must be used as input operands later in the computation. The problem arises when an intermediate result that is needed as the input to the next operation is not available, but is instead stuck somewhere in the arithmetic element's pipeline. The user must solve this problem by either allowing time for the intermediate result to fall out of the pipe, thus reducing throughput, or by overlapping other operations, thus complicating system programming and control.

Architectural implications for system design

When creating a digital signal processing system, a designer can choose one of two approaches to hardware implementation. The first approach is to base the design on a single-chip digital signal processor, choosing from a small but steadily-growing number of product entries; a typical single-chip digital signal processor architecture is shown in fig. 2. These processors are usually relegated to lower-speed applications, though the meaning of the phrase "lower-speed" is changing almost daily. The second approach is to use a multi-chip solution, designing the system with arithmetic/control building blocks configured in a manner that best suits the application. A typical multi-chip array processing system based on the Am29500 series of digital signal processing parts is shown in fig. 3.

While single-chip processors are becoming more sophisticated and powerful, they are still, in most cases, not capable of handling the more demanding digital signal processing applications. The multi-chip solution gives the user the ability to optimize the architecture, adding data paths and arithmetic elements as needed. The multi-chip solution in fig. 3, for example, is capable of performing a 1024-point complex-input FFT in 2 milliseconds; if a user must perform an FFT in less time, the necessary arithmetic elements and data buses can be added to increase throughput.

If the application is such that only a multi-chip solution is feasible, it is crucial that attention be given to minimizing signal delay between communicating components. Bipolar devices have the advantage in such systems, because their higher gate speeds and superior drive capability significantly reduce chip-to-chip delays. Designers building multi-chip solutions from CMOS must take particular care to assure that communication between components is fast enough to keep pace with the processing power of those components.

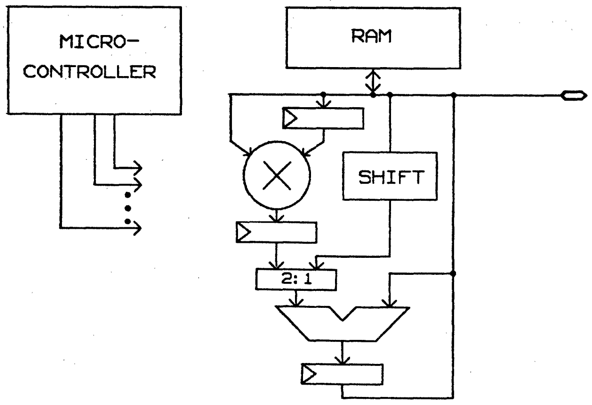


Fig. 2: Typical single-chip digital signal processor architecture.

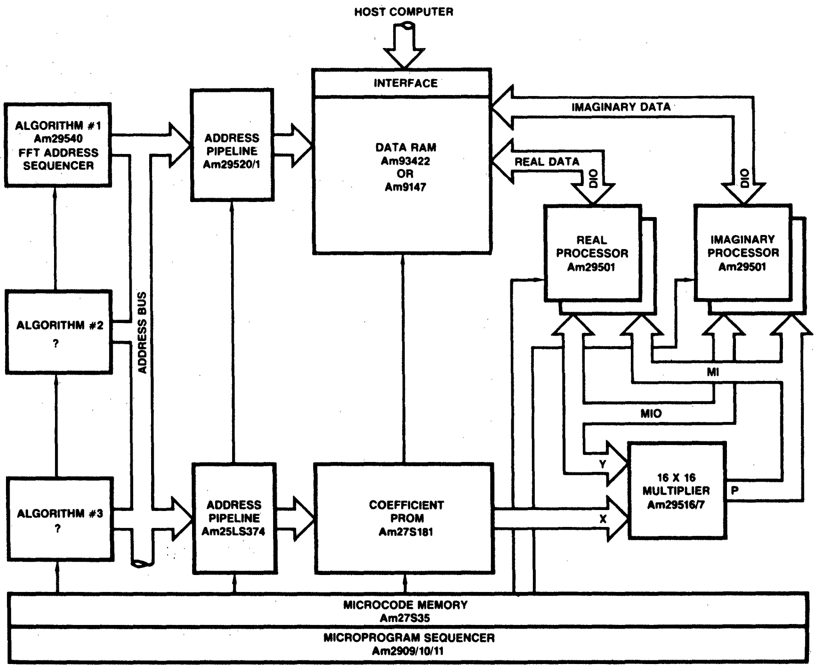

HOST COMPUTER
INTERFACE
IMAGINARY DATA
ALGORITHM #1 Am29540 FFT ADDRESS SEQUENCER
ADDRESS PIPELINE Am29520/1
DATA RAM Am93422 OR Am9147
REAL DATA
DIO
DIO
REAL PROCESSOR Am29501
IMAGINARY PROCESSOR Am29501
ALGORITHM #2 ?
ADDRESS BUS
MI
MIO
Y
16 X 16 MULTIPLIER Am29516/7
P
X
ALGORITHM #3 ?
ADDRESS PIPELINE Am25LS374
COEFFICIENT PROM Am27S181
MICROCODE MEMORY Am27S35
MICROPROGRAM SEQUENCER Am2909/10/11

Fig. 3: Typical multi-chip digital signal processor architecture, using the Am29500 DSP family.

4/3

Conclusions

There is little doubt that CMOS technology is a worthy competitor to bipolar, and that its infuence and capabilities will grow as time goes on. CMOS brings moderate-to-high performance digital signal processing to applications whose lower power and cooling requirements preclude the use of bipolar technology.

It must be stressed, however, that the reports of bipolar's death are greatly exaggerated. Bipolar still offers higher speeds and greater architectural flexibility than CMOS, and will continue to do so for the forseeable future.

Bibliography:

Downing, P., Gwozdz, P., and New, B. "Denser Process Gets the Most Out of Bipolar VLSI," **Electronics**, June 28, 1984, pp. 131-133.

Bursky, D. "CMOS Processes Shrink Geometries, Pick up Speed for RAMs, Multipliers," **Electronic Design**, Vol. 32, No. 9, 1984.

A HIGH PERFORMANCE CMOS PROCESS FOR THE NEXT GENERATION EPROM

Jih Lien, Steve Longcor, K.Y. Chang, Lewis Shen,
Pete Manos, David Chan, Tien Lee, and Sucheta Nallamothu

Advanced Micro Devices
Sunnyvale, CA 94088

ABSTRACT

A high performance CMOS process has been developed for the fabrication of the next generation EPROM family. The effective channel length of the EPROM transistor is approximately 1.0um, with the gate and inter-poly oxide thicknesses, and junction depths scaled accordingly. Using this transistor, the cell size can be scaled to less than 20um^2. By utilizing state-of-the-art lithography, dry etch, and glass reflow technologies, a one megabit EPROM can be fabricated. With proper shrinking of the dimensions, the cell can be reduced to approximately 12um^2, demonstrating that this process is sufficient for future EPROM's in the multi-million-bit range.

INTRODUCTION

Recent advances in NMOS technology have brought EPROM manufacturing to the 512Kb level [1]. An experimental NMOS 1Mb EPROM has also been presented [2]. The high power consumption of NMOS circuits limits the application of very high density NMOS memory chips in future generation microprocessors and computer systems. On the other hand, CMOS circuits have the advantage of low power consumption and high noise margin, making them especially suitable for high density design. Furthermore, as the feature size shrinks, the speed performance of CMOS circuits begins to improve. At the megabit level the speed of a CMOS EPROM is comparable with or even superior to its NMOS counterpart.

Latch-up susceptibility has been a basic problem for CMOS circuits, which becomes even more serious when high voltages and currents are present. Using a p^- epitaxial layer on a p^+ substrate can increase the holding current significantly, but there are still several avalanche-induced latch-up triggering modes which cannot be corrected by using epi wafers. Design and layout techniques are used to remedy these problems and ensure safe operation of the circuits.

In this paper, we will describe our CMOS process, show memory device characteristics, and present methods of preventing latch-up triggering.

FABRICATION TECHNOLOGY

Primary considerations for the development of this CMOS process were: (a) maximum compatability with present NMOS EPROM processes, (b) suitability for high-volume production, and (c) shrinkability for future multi-million-bit level designs. An n-well process is the logical choice for an EPROM product and allows the desired NMOS compatibility. The p^- type starting material has the same resistivity as the previous generation NMOS process.

Table 1 shows the key features of this process. A conventional LOCOS isolation scheme is used, with the field oxide thickness scaled for small cell size. The EPROM transistors fabricated in this process are conventional double-poly stacked-gate structures. They are derived from proven NMOS EPROMs by appropriate scaling of device dimensions. An anisotropic dry etch process is important for the definition of this double-poly structure. Figure 1 shows a cross-sectional view of the stacked-gate structure. Figure 2a shows the top view of a portion of the memory array using a cell of approximately 20um^2. Figures 2b and 2c are similar views of the 90% (16um^2) and 80% (12um^2) reductions respectively. A low temperature glass reflow module is used to improve the metal step coverage over contacts. An anti-reflection coating is needed for improved metal line definition.

A 21-stage ring oscillator with a fan-out of one has been used for the evaluation of this process. At V_{DD} = 5V the gate delay is 280 ps and the corresponding speed-power product is 0.031 pJ.

MEMORY TRANSISTOR CHARACTERISTICS

The EPROM transistors described above have been fully characterized under read and program conditions. Device sizes suitable for memory cells as small as $12um^2$ have been studied. Figure 3 shows one-shot I_{DS} vs. V_{DS} curves during programming for the 100% and 80% sizes. Both curves indicate critical programming (drain) voltages of less than 5 volts: they could be programmed using V_{DD} = 5V. Instead of sacrificing programming speed to achieve 5 V-only operation, optimal voltages are applied to both control gate and drain. Figure 4 clearly shows the importance of drain voltage in determining ΔV_T in the 0-5ms time range. An on-chip voltage pump provides a gate voltage larger than V_{PP}. The voltages are regulated by circuit techniques to prevent damage to the memory transistor, selecting programming in the hot-electron injection region rather than in the impact ionization breakdown region.

As can be seen from Figure 3, drain voltages of 7V for the 100% version and 6V for the 80% version are safely below the breakdown region. Punchthrough voltages (with V_G = 0.2V) for unprogrammed EPROMs are > 7.5V for the 80% version and > 10V for the 100% version. Figures 4 (100%) and 5 (80%) show that the desired programming speeds have been achieved under these conditions. The optimum gate voltage can be determined by measuring programming speeds or by studying gate current vs. floating gate voltage curves (see Figure 6). Peak injection current results from a V_{FG} near 10V, corresponding to a control gate voltage of approximately 14V. Note that during actual programming, the gate current will rapidly approach zero as charge collects on the floating gate. The drain voltage supplied by regulating circuitry will also rise somewhat as programming proceeds.

Given the similarity between the EPROM transistors produced by this process and their proven NMOS counterparts, high product reliability should be expected. In particular, measured charge loss characteristics are similar to 512K NMOS EPROM transistors (which demonstrate data retention far in excess of ten years).

LATCH-UP PREVENTION

High voltage operation during programming requires special care in process development, circuit design, and layout to prevent the occurence of latch-up. Figure 7 shows the holding current as a function of p^+ to n^+ spacing. The use of p^- epi on p^+ substrate offers significantly higher holding current for large p^+ to n^+ spacing, but is ineffective at small spacing. Epi is useful only when proper layout rules are followed in the design. The effect of n-well implant drive time is also shown. In order to minimize n-well side diffusion, thinner epi is generally preferable to longer well drives for achieving the desired relationship between well depth and p^- layer thickness.

Epi wafers provide higher holding current and prevent the CMOS circuits from permanently latching-up, but do not prevent the occurence of triggering (transient latch-up). On the other hand, epi wafers are more vulnerable to V_{SS} line noise and gate noise because the lateral NPN transistor has higher current gain. Triggering does not always damage the device but causes malfunction of the circuit.

An inverter can be triggered by substrate current from a neighboring n-channel transistor even at a distance of 400um. During programming, the total substrate curent generated by the EPROM array can be as high as 1-2 mA, which poses a danger of triggering nearby peripheral circuits. Double guard rings are provided as current sinks around the array. The inner p^+ guard ring collects substrate current. The outer n^+ guard ring serves as a second collector to divert electrons that would otherwise flow into the n-well. A 50% reduction in substrate current was observed by using double guard rings.

As a second protection, guard rings were also applied around selected peripheral circuits to avoid possible triggering induced by substrate current. An n^+ guard ring around the p-channel transistor increases the NPN mode triggering current significantly and is more effective in preventing substrate current-induced triggering than a p^+ guard ring around the n-channel transistor (see Figure 8).

SUMMARY

A high performance CMOS process has been described. Fabrication of EPROM arrays and characterization of the memory cell transistors demonstrate shrinkability suitable for multi-million-bit level circuits. Latch-up susceptibility of CMOS circuits can be prevented by using suitable design and layout techniques. The simplicity of this process makes it compatible with volume production.

ACKNOWLEDGEMENT

The authors would like to acknowledge the support of John Stanczak in test chip design and layout, Al Miller in electrical measurement, Yasmin Nawaz in photolithography development, and the Fab VII staff in processing the wafers. Consultations with the EPROM design group during the preparation of this paper are highly appreciated.

REFERENCES

[1] D. Rinerson, M. Ahrens, J. Lien, B. Venkatesh, T. Lin, P. Song, S. Longcor, L. Shen, D. Rogers, and M. Briner, "512K EPROMs," ISSCC Digest of Technical Papers, Feb. 1984, p. 136.

[2] K. Okumura, S. Ohya, M. Yamamoto, T. Watanabe, Y. Shimamura, and M. Kikuchi, "A 1Mb EPROM," ISSCC Digest of Technical Papers, Feb. 1984, p. 140.

Table 1. Process Features (100% Version)

silicon gate n-well CMOS
two polysilicon layers

L_{eff} (min.)	1.0 um
W_{eff} (min.)	0.9 um
gate oxide	250 A
interpoly oxide	300 A
field oxide	5500 A
X_j (n^+)	0.2 um
X_j (p^+)	0.35 um
X_j (n-well)	3.5 um
V_t (n)	0.6 V
V_t (p)	-0.9 V
V_t (EPROM)	1.0 V
speed-power product	0.031 pJ
gate delay	280 ps

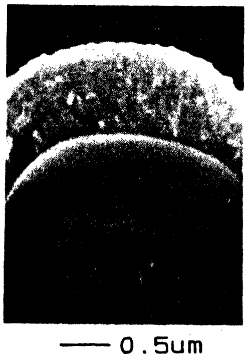


Fig. 1. SEM cross section of the stacked-gate structure.

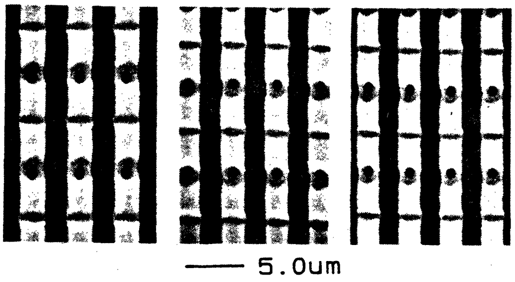


Fig. 2. Top views (left to right) of the (a) 100%, (b) 90%, and (c) 80% EPROM arrays.

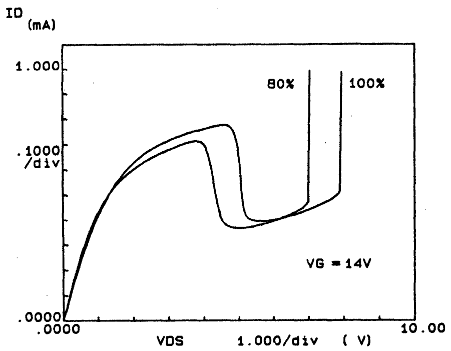


Fig. 3. One-shot EPROM characteristic during programming.

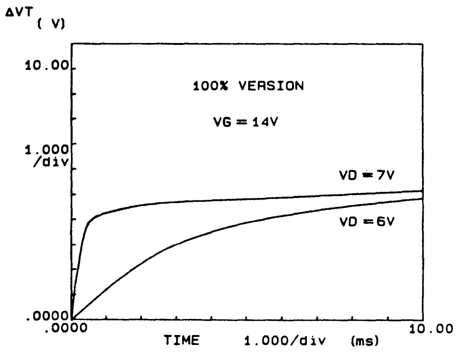


Fig. 4. V_T change vs. programming time (100% version).

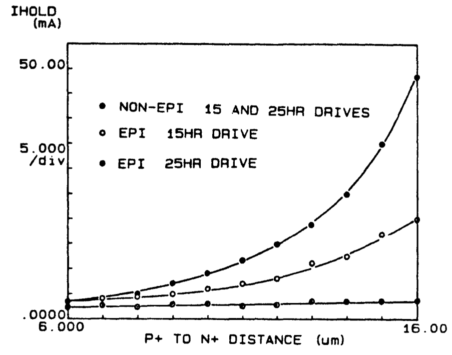


Fig. 7. Holding current vs. p^+ to n^+ distance.

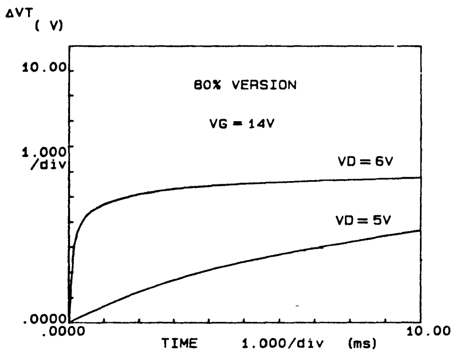


Fig. 5. V_T change vs. programming time (80% version).

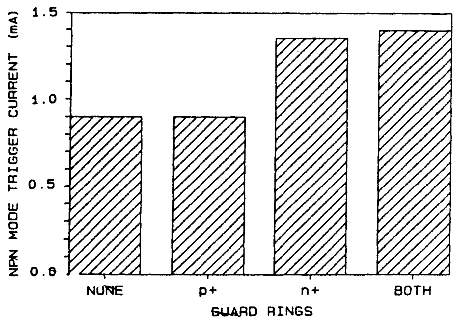


Fig. 8. NPN triggering current for different guard rings.

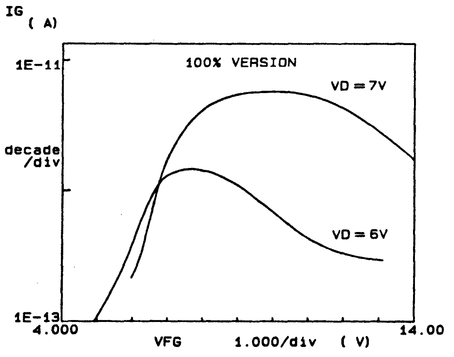


Fig. 6. Gate current vs. floating gate voltage.

Surface Mount Technology, Plastic Chip Carrier Trends

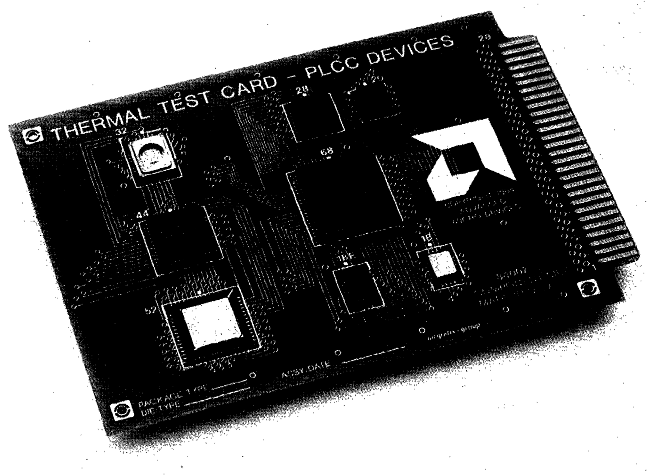
By Michael C. Lancaster, Manager, Advanced Package and Materials Development Group, Advanced Micro Devices, Inc., Sunnyvale, Calif.

The word is being spread all over the world through well-attended seminars and articles in trade journals that surface mount technology is the packaging revolution of the 1980s. By now most engineers and managers under the electronics umbrella know of the reasons and need for surface mount technology. The impact to systems houses of increased board density, elimination of costly through-holes, fewer board layers, and the ease of automation of populating boards have been well publicized. So why this meteoric interest and demand?

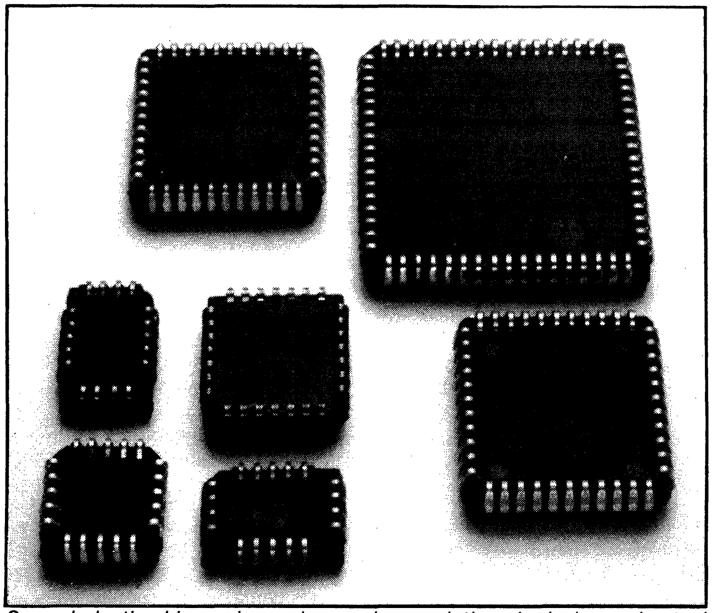
In the passive component field, surface mount resistors and capacitors have been available for a decade, most notably in Japan. Driven by consumer-oriented products, the need to reduce product size, weight, and cost has been of paramount importance. Since these products primarily consist of analog circuitry, treating radio frequency (RF) signals instead of digital codes, resistors, and capacitors are required in great numbers in resistor-capacitor inductance networks. It is estimated that more than 30 percent of the Japanese consumption of passive components (63 billion units) is already of surface mount (SM) configuration. In addition, because most Japanese companies are vertically integrated, surface mountable IC packages have been developed for inclusion in about 16 percent of their products. However, this approach resulted in a proliferation of outlines with no attempt at standardization.

When a need for surface mount components was identified in the United States, manufacturers of passive components quickly responded to avoid orders going overseas. Standards were developed for component values and physical size. Today a full range of surface mount resistors from 10Ω-2.2MΩ of one size (1.6 × 3.2 × 0.6mm) and surface mount capacitors 680 pF-0.5μF are readily available. The lack of standardization and the horrendous capital investment required by semiconductor suppliers to tool and internally handle SM configurations have resulted in a reluctance to "go it alone." Thus frustration from lack of availability of SM IC's has been building as total conversion to surface mount technology has been severely hampered. Companies such as Delco have played a critical part in forcing a standard between major suppliers and in turn an industry-registered family outline.

Developed out of Texas Instruments' initial design concepts, standard outlines are in the final stages of approval by the Joint Electronic Device Engineering Council (JEDEC). With standards in place and demand from computer and telecommunication industries for SM components now being added to these initial consumer applications, the rush for SM IC availability and secured source is on.



Components surface mounted on a thermal test board.


Several plastic chip carrier packages show variations in design and actual mounted capability.

IC Package Availability

The main and earliest motivation for surface mount technology in IC packaging has come from government contracts. In fact, the long standing flatpack which appeared in the late 1950s is still widely used in military applications, utilizing surface mount technology on ceramic substrates. The three contracts awarded in the late 1970s to RCA, Hughes, and Texas Instruments to develop leadless chip carriers was the first major thrust to find an alternative to the flatpack that would satisfy MIL-Std-883 and 38510. The family of packages developed served as a useful impetus to piece-part and socket manufacturers, test handler suppliers, and users to set industry standards of package design and PCB interface.

As a result, most semiconductor suppliers now offer leadless chip carriers as a package option to their full range of products, and thus, the first widely available surface mountable package.

Despite the availability of this surface mountable family, the expected rush of orders did not materialize. This could be attributed to two things. First, the packages were two to three times more expensive than their equivalent ceramic dual inline packages and five to six times more than the plastic package. Furthermore, the expected cost reduction to the single layer and metallized package (SLAM) never materialized because of the unreliability of the cap seal.

Second, the thermal mismatch between the ceramic piece-part and a polyimide or epoxy PC weakened the solder joints and was found to be an unacceptable method of attachment for lead counts of 44 and above. For these applications, sockets were needed which negated surface mount technology and added approximately one dollar to the cost of each part.

For 48 leaded parts and above, however, where there was not an acceptable dual in-line package (DIP) alternative to the multilayer packages, these ceramic chip carriers have found a niche in the marketplace. The superior electrical parameter performance to be achieved with a leadless chip carrier (LCC) and the lower cost than multilayer DIPs have outweighed the additional socket costs. However, even this market has been eroded recently with improved plastic DIP processing techniques up to 64 lead devices.

The salvation for this SM requirement with thermal compliance to a variety of substrates has been to sacrifice hermeticity and accept a plastic package.

PLCC and SO Packages

Two plastic surface mount package configurations have been considered. These are the plastic leaded chip carrier family, also known as a PLCC, PCC, or quadpak, and the small outline (SO) family. To understand the development of these packages, the derivation of the outline history needs to be expounded.

The oldest and therefore more established package of the two is the SO. Originally developed in Europe by Philips, the SO package (or Swiss Outline as it was termed, displaying its European origin) found first application in discrete devices also adopted by Motorola. As the surface mount requirements in consumer and telecommunications industries mirrored the Japanese approach, IC's with 14 and 16 leads were packaged in SO. These had a 150 mil wide body with dual in-line leads with individual spacing of 50 mils with the leads bent in a gull wing fashion or a butt joint. The package size of the SO was approximately half its DIP counterpart. Recent trends have been to extend lead counts up to 28 with increased body widths up to 300 mils (table 1).

Texas Instruments has been the chief propounder of the plastic quad package. Originally developed for internal use, the package has found general acceptance in the industry. With few modifications, JEDEC recently accepted registration of the square package family and may soon be accepting the rectangular family.

PLCC, a late-comer compared to the SO package, has made significant in-roads in establishing itself as the standard. The configuration lends itself to accommodate very large scale integrations (VLSI) die with high lead counts and, perhaps more importantly, is compatible with the footprint established for LCC. Initial reliability data shows it to be comparable with the DIP with potential for improvement.

When determining the properties of the leadframe material to be used in DIP's or SO's, a compromise needs to be reached between a strong frame to minimize lead damage and a frame with good thermal properties. Unfortunately, for metals, the higher the thermal conductivity, the lower its mechanical strength (table 2). Since with PLCC the lead is wrapped under and locked in place under the package, there is little concern for subsequent lead damage and bending. Hence softer copper materials which have improved thermal characteristics can be used for the leadframe. A further consequence of utilizing a softer frame material is that the leadframe plastic pullaway created during the lead forming operation is minimized. Recently, questions have arisen concerning the reliability of the SO package because of the thin plastic body and narrow path from die to package body outline. This has been hotly contested by SO manufacturers[3]

There is an obvious need for the SO package, which will become increasingly popular for lead counts below 20. For more than 28 leads, it is clear the PLCC is presently the only viable SM package with at least a DIP reliability. There will be an overlap of technologies for lead counts of 20, 22, and 24. For semiconductor companies with large volumes of low lead count parts, it is in their interest to promote the SO package to as high a lead count as a marketplace is found. Companies that supply primarily high lead count products will extend the PLCC family to as low a lead count as there is a market (table 1).

Both Philips at their Surface Mount Technology Center in Milwaukee and Texas Instruments at their facility in Houston offer extensive surface mount technology training courses and capabilities for prototype SM on boards, naturally biased to SO and PLCC technologies, respectively.

The dynamic random access memory (DRAM) market does not know whether to go quad or SO, and many companies are sitting poised to go either way depending on which way the big computer companies commit. For memory arrays, a dual in-line configuration is still the best for board routing and interconnection of the discrete memory modules. Quad packaging would force systems houses to another board layer (three instead of two), thus adding to costs. Texas Instruments proposes to adopt a modified SO package utilizing the "J" bend concept of the quad package and is being considered for their 256K and a further modified SO, but still 300 mil, for their proposed megabit. Here they are in conflict with Intel, which favors a 350 mil wide × 550 mil PLCC[4]

High Lead Count Direction

Is there a surface mountable package for lead counts in excess of 100? To partially answer this question, consider the package area versus lead count (figure 1). For lead counts of less than 100, the 50 mil lead-spaced PLCC shows the most effective use of board space, but for more than 100, the pin grid array (PGA) package is more efficient. Although this is not a surface mount part, it is already an established industry standard.

If the lead spacing on a PLCC is reduced to a 25 mil pitch, the PLCC immediately becomes the most attractive configuration, and furthermore, is compatible with surface mount technology. Before this configuration can be introduced, systems houses have to agree that they can accommodate wave or reflow soldering in pads of 25 mil pitch. One of the largest computer manufacturing companies recently announced that this is now possible. However, a further limitation is set by the semiconductor manufacturers and piece-part vendors.

For a 120-lead device, the leadframe stamping and discrete wire bond constraints on finger width and spacing dictate that the leads can converge toward the package center only to an envelope 500 mils square. This may be too large to accommodate die, especially since wire sweep could be a serious issue with the PLCC and wire lengths needing to be minimized.

An alternative would be to reduce frame thickness, which would make possible narrower fingers and spacing. The full implications of accomplishing this, however, need to be investigated. As for the military requirements for surface mountable hermetic packages, it is expected that an extension of the existing

TABLE 1 — SO AND PLCC CONFIGURATION VERSUS LEAD COUNT

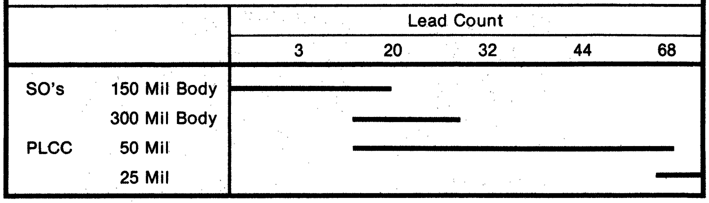


TABLE 2 — CONDUCTIVITY, STRENGTH AND RESULTING PACKAGE θja FOR DIFFERENT MATERIALS AND PACKAGE CONFIGURATIONS

	Conductivity BTU•Ft/Ft²•HR•°F	Tensile Strength ¾ Hard	θja°C/W			
			40L DIP	48L DIP	64L DIP	68 PLCC
Alloy 42	8.8	92	66	72		
Olin 195	110	76-84	44	55		
Olin 194	142	57-67			35	
Olin 155	189	48-58				
Olin 151	197	47-56				37

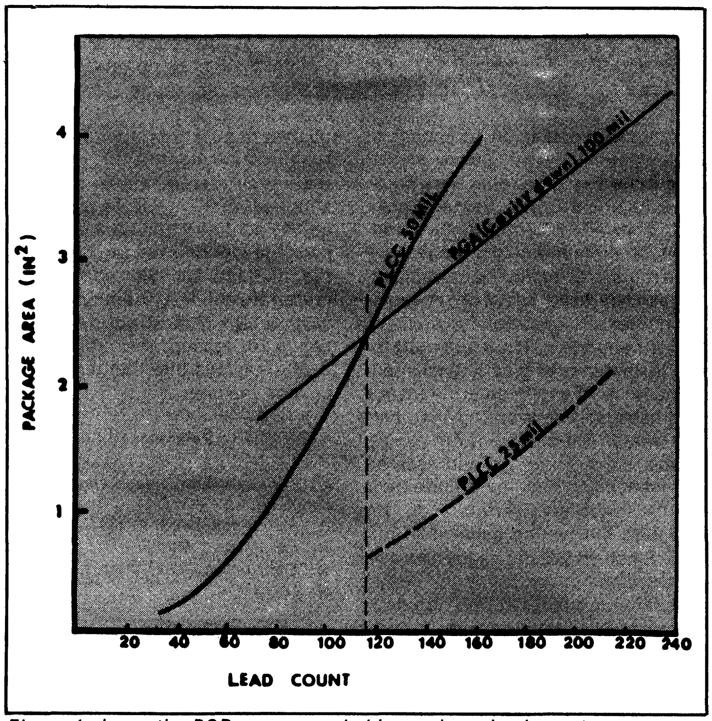


Figure 1 shows the PCB area occupied by package lead count.

leadless chip carrier family will be developed with either 25 or 20 mil pad pitch to satisfy the high lead count needs (table 1).

Talk of yet a lower pitch, 12.5 mils, has been suggested but will probably have to wait until the applications of the TAB technology is universally accepted after implementation in high lead count and VLSI fields. System houses may also have to develop alternatives to wave soldering.

Response to Demand

There has been hesitation on the part of semiconductor device suppliers in the United States to join the surface mount technology bandwagon and support their customer needs. They have been content to continue packaging IC's in the DIP format and develop another through-the-hole packaging concept (PGA) for higher lead count devices where the DIP format is impractical.

A notable exception to this has been Texas Instruments. Because of a captive application for their devices and the tight control over their packaging and interconnection technology, they have been able to develop the quad package through to production independent of other system houses.

The majority of semiconductor device manufacturers do not have captive applications and supply electrical components to the consumer, telecommunication, and computer markets. The reluctance to commit to surface mount technology has been largely due to a lack of industry standards. These are close to becoming corrected, and most companies are in a position to eagerly respond to surface mount technology demands. AMD, for instance, has anticipated the standards being approved and is well under way with the qualification procedures. The initial thrust has been to establish qualified subcontractor capability, but to provide in-house capability for expected high volume runners (44 and 68 lead) or when subcontractor tooling was not readily available (as in the case of 18 lead rectangular).

The requirements for tooling unfortunately has hit tool manufacturers just at the time when business was expanding because of a worldwide increase in demand for standard DIP packages. Thus an already desperate situation was exacerbated, pushing mold leadtimes out to 30-40 weeks.

Subcontractor assembly houses have been quick to see the potential increase in their business by supplying PLCC capabilities. In fact, not only have they offered assembly capability, but also leadframe availability. In the scramble for PLCC product capability, heavy demand will be placed on this, and availability of capacity will go to companies that have supported this tooling or have partaken in early production volumes.

Why have AMD and semiconductor device manufacturers taken this and similar approaches to surface mount technology? First, the cost savings to be obtained in converting are all at the level two of packaging. This is the systems board level and hence the major incentive is not with the supplier. This systems-driven need resulted in uncertainty on the suppliers' part as to the seriousness of their vendors, the type, and industry uniformity of desired configurations.

Second, manufacturers have been pushing for an industry-wide agreement on standards before committing significant amounts of capital investment in tooling. To appreciate fully the need for standardization, consider for a moment the costs involved in providing even a modest capacity in PLCC. New mold and deflash, trim, and form equipment have to be purchased for each lead count, and both die attach and wire bond indexing carriages have to be modified. In the test area, burn-in sockets, electrical test sockets, and environmental handlers have to be purchased. This total investment in capital per lead count could exceed $400,000. As for leadframe tooling, an initial etched leadframe source can be established for approximately $20,000, but if volumes explode as some pundits predict, a stamping source must be established at a cost of $120,000 per lead count.

Unfortunately, unlike DIP configurations in which one stamping tool can offer several cavity size options, the tight tolerances and geometry design considerations for PLCC frames dictate that one stamping tool only gives one cavity size. Some stampers will offer two, but this is the maximum, and some compromise in design is anticipated. Thus not only do leadframes have to be tooled for 10 different lead counts — 20, 28, 44, 52, 68, 84 square, and 18, 22, 28 and 32 rectangular — but maybe two or three cavity sizes as well. This in itself necessarily imposes international standardization and unprecedented cooperation between semiconductor device manufacturers and/or with leadframe suppliers to share this astronomical tooling cost, estimated at $5 million.

Component Demand

Four years ago, the expected demand for LCC never materialized. Will the same thing happen this time? Perhaps not, for the reasons discussed below.

The LCC revolution was driven by military markets, and the cost reduction, which was expected to attract other marketing outlets, never materialized. This time demand is emanating from major computer and telecommunications system houses driven by a tremendous need to reduce size and cost of their system products. It has been estimated[1] that 50 percent of the worldwide production of IC's is consumed in the United States, and the greatest users by far are companies with interests in the above areas. IBM (Data Entry Systems Div.), Digital Equipment Corp., Hewlett-Packard, and AT&T all have declared their determination to pursue surface mount technology, and in many cases they have sent out requests for quotes.

Furthermore, a significant shift to the acceptance of plastic packaged devices in the computer industry has been observed during the past two years. The quality of die passivation, mold materials, and assembly processing have all contributed to the excellent performance to moisture life testing achievable today. Since PLCC packages are assembled using identical technology, full advantage can be gained through these already proven advances in quality levels.

Semiconductor component manufacturers are now responding aggressively to the SM demand Frustration of component availability will not disappear, however, as manufacturers scramble for limited available capacity and places in line at their vendor tool manufacturers. Within 12-18 months, it is predicted that most device types presently supplied in DIP's will be offered in an SM configuration as a customer package option. Let us hope that the demand for SM components materializes and semiconductor component manufacturers can begin to recoup the tremendous capital outlays that this technology change requires. **E•O**

References

1. D. Rose, Rose Associates, private communication.
2. "E.E. Times," April 9, 1984, p. 61.
3. "E.E. Times," May 7, 1984. p. 1.
4. "Electronics Week," August 13, 1984, pp. 12-13.

Contact author at Advanced Micro Devices, Inc., 901 Thompson Place, P.O. Box 3453, Sunnyvale, Calif. 94088. Telephone: 408/732-2400.

Electronique, Techniques et Industries (Paris), 1984, nº 8, pp. 29-36

La conception d'une unité centrale microprogrammable

D. Mithani

Advanced Micro Devices.
901, Thompson Place, Sunnyvale, CA 94088

La microprogrammation est de plus en plus utilisée pour réaliser la logique de contrôle de systèmes allant des microprocesseurs aux ordinateurs universels. Comparés aux blocs VLSI microprogrammables par l'utilisateur, les microprocesseurs classiques comportant des systèmes microprogrammés incorporés ne présentent pas une grande souplesse. L'arrangement de blocs microprogrammables par l'utilisateur permet aux concepteurs d'adapter l'architecture des systèmes à diverses applications. De plus, il est plus facile d'améliorer les performances du système en implantant les programmes les plus souvent utilisés (compilateurs et traitement de texte par exemple) dans la mémoire du microprogramme plutôt qu'à l'intérieur de la mémoire centrale. Une architecture de système peut également être utilisée pour des applications différentes en changeant le contenu de la mémoire du microprogramme.

Introduction

C'est un système microprogrammé conçu avec la famille Am 2900 bien connue, que nous traitons dans cet article. Au centre du système figure le microprocesseur bipolaire 16 bits Am 29116 *(fig. 1)*. En raison de son architecture et de son jeu d'instructions, ce composant convient aux unités centrales à performances élevées. La description présentée ici essaie de conserver la compatibilité du logiciel de l'architecture avec le Super-16 bien connu, ordinateur 16 bits conçu avec des Am 2903 en tranche de 4 bits.

L'architecture de ce microprocesseur est en pipeline aussi bien au niveau du microprogramme que des macroinstructions. Son jeu d'instructions est identique à celui du Super-16 ; il peut donc traiter le logiciel du Super-16 sans aucune modification. Bien que les performances optimales de l'Am 29116 soient obtenues pour les contrôleurs de périphériques, il est idéal pour cette unité centrale. Son jeu d'instructions lui permet d'exécuter des opérations arithmétiques, des transferts de données, des décalages de plusieurs bits,

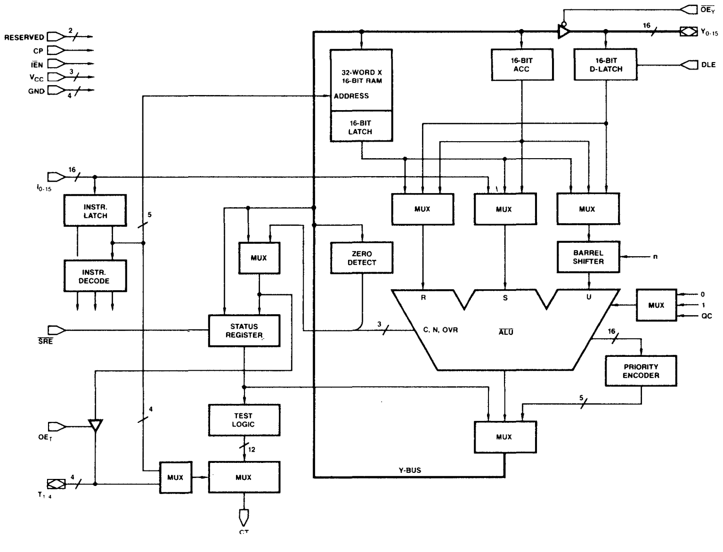


Fig. 1 — Architecture interne de l'Am 29116.
— *Detailed bloc diagram of Am 29116.*

Designing a microprogrammable CPU

Microprogramming is becoming increasingly popular in implementing control logic in everything from microprocessors to large main frames. Conventional microprocessors with built-in microprogrammed systems do not provide much flexibility when compared to user-microprogrammable VLSI building blocks. These user-microprogrammable building blocks allow designers to customize system architecture for various applications. In addition, it is easier to upgrade system performance by embedding most commonly used software routines (compilers and word processors, for example) in the microprogram memory than in the main memory. Also, one system architecture can be used for different applications by changing the contents of the microprogram memory.

des manipulations de bits et d'états. Outre sa rapidité, l'organisation de l'Am 29116 réduit la consommation de courant ainsi que la surface de circuit imprimé.

Organisation du système

Dans un système simple comprenant une mémoire centrale de 16 bits de large et l'unité centrale *(fig. 2)*, la mémoire centrale est composée de RAM statiques (telles que Am 93422). Une structure à bus unique assure la communication entre les deux sections. Bien que l'on ne traite pas de l'interface avec d'autres dispositifs d'entrée/sortie, on peut facilement modifier le bus afin qu'il prenne en compte : la demande de bus, l'accusé de réception et d'autres signaux d'interface. Si on ajoutait d'autres dispositifs d'E/S, il faudrait soit un contrôleur de bus qui assure l'arbitrage des demandes de bus provenant de l'unité centrale ou des dispositifs d'E/S nécessitant des transferts DMA (Direct Memory Access).

La *figure 3* décrit les signaux d'interface échangés entre la mémoire et l'unité centrale. La transmission est effectuée par un bus d'adresses sur 16 bits, un bus de données bidirectionnel sur 16 bits et un bus de commandes. Celui-ci comprend la demande de mémoire, lecture/écriture, reconnaissance d'adresse, échantillonnage de données, synchronisation des données et des lignes de contrôle d'interruption.

Afin d'utiliser les données pendant le cycle n+1, l'Am 29116 génère l'adresse mémoire centrale pendant la cycle n-1 et les données sont lues pendant le cycle n. Le n^e cycle est plus long que les cycles normaux de l'unité centrale afin de pouvoir tenir compte des temps de lecture de la mémoire centrale. Les informations qui allongent le cycle sont fournies au généra-

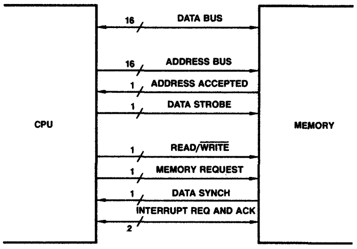


Fig. 3 — Protocole d'échange entre mémoire et CPU.
— *Memory - CPU handshaking protocol.*

teur d'horloge du système pendant le cycle (n-1)^e. L'unité centrale génère l'adresse, la demande de mémoire et les signaux de lecture/écriture pendant le cycle n. La mémoire centrale répond aux demandes de la mémoire en plaçant le signal de reconnaissance d'adresse à l'état BAS. Si la mémoire est occupée, elle ne génère pas ce signal et l'unité centrale attend alors de le recevoir. L'unité centrale répond au signal de reconnaissance d'adresse en générant le signal d'échantillonnage des données, ce qui indique qu'elle attend des données en provenance de la mémoire. La mémoire génère le signal de synchronisation des données, ce qui indique au bus que les données sont validées pour l'opération de lecture. Les signaux d'échantillonnage des données et de synchronisation des données génèrent soit le signal de chargement du « latch Z », soit le signal de chargement du registre de données, selon que l'information demandée est une instruction ou une donnée.

Puisque l'Am29116 génère

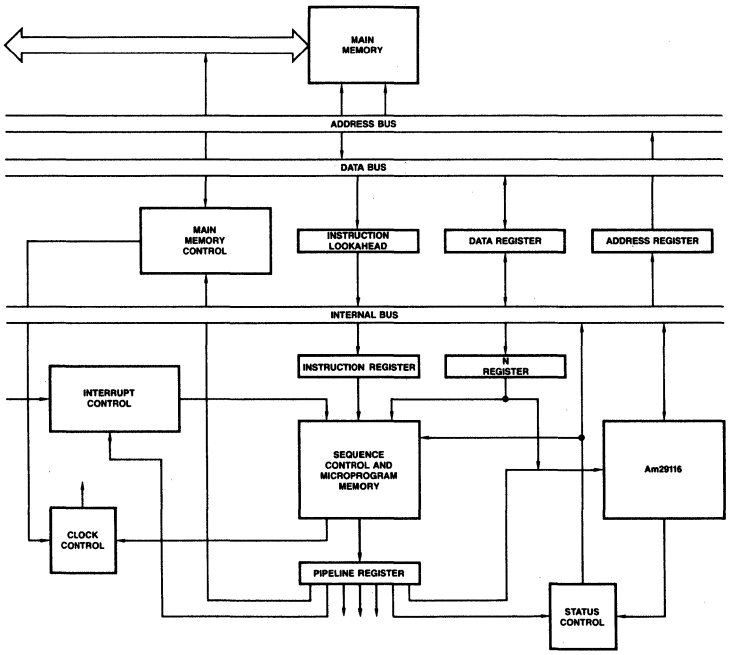


Fig. 2 — Architecture de l'unité centrale.
— *Central processing unit block diagram.*

l'adresse mémoire centrale et les données qui doivent être écrites, l'opération d'écriture en mémoire est effectuée en deux cycles. L'adresse et la demande de mémoire sont maintenues actives pendant deux périodes d'horloge. La période n'est pas plus longue que la période normale de l'unité centrale afin de pouvoir tenir compte des temps d'écriture de la mémoire. Si la mémoire n'est pas occupée, le signal lecture/écriture sera généré au cours du prochain cycle en même temps que le signal d'échantillonnage des données, indiquant la validité des données à écrire sur le bus. L'impulsion d'écriture de la mémoire centrale est générée à partir des signaux d'échantillonnage de données et de lecture/écriture. C'est l'impulsion de synchronisation des données qui indique à l'unité centrale que l'écriture des données est acceptée.

Format des instructions

Les intructions stockées en mémoire centrale ont une longueur d'un ou deux mots, un mot ayant une largeur de 16 bits *(fig. 4)*. La moitié de poids fort du premier mot (MSH) constitue le champ du code opératoire. La moitié de poids faible (LSH) divisée en deux champs de 4 bits chacun, indique quel est le registre affecté aux opérandes et au résultat.

L'utilisateur se sert de la moitié inférieure (registre 0 à 15) du bloc de 32 registres de l'Am 29116 comme « brouillon ». Le système d'exploitation utilise la moitié supérieure (registres 16 à 31) pour conserver une trace des piles de mémoire, du compteur, etc. Le premier mot d'une instruction à deux mots possède un format identique à celui de l'instruction à mot unique décrite ci-dessus. Le second mot a toujours une valeur de 16 bits, qui est soit une adresse de déplacement, soit une donnée immédiate. Un code opératoire de 8 bits de large permet 256 instructions, ce qui suffit amplement à un ordinateur universel. Le jeu d'instructions comporte des codes opératoires qui peuvent travailler sur les types de données suivantes : bit, quartet, octet, mot.

Les informations sur le mode d'adressage contenues dans le code opératoire peuvent s'appliquer à des instructions différentes. En outre, le jeu d'instructions comporte l'instruction PUSH/POP qui conserve les piles simples ou multiples, les instructions d'entrées/sorties et les calculs arithmétiques sur les nombres entiers binaires et décimaux.

Selon le mode d'adressage, le registre spécifié par l'instruction peut être soit un accumulateur pour des opérations logiques et arithmétiques, soit un registre index servant à manipuler l'adresse de l'opérande en mémoire centrale. Pour les opérations dont le résultat est chargé dans le registre, le champ R1 indique l'adresse du registre destination et R2 (ou R2 + d) constitue le registre source (ou bien pointe vers le champ source dans la mémoire centrale). Pour les opérations dont le résultat est transféré du registre à la mémoire centrale, le champ R1 indique l'adresse du registre source et R2 (R2 + d) indique l'adresse destination en mémoire. Pour le transfert de mémoire à mémoire, le champ R2 constitue le pointeur source et R1 est le pointeur de destination pour la mémoire centrale.

La souplesse de l'architecture du microprogramme permet aux concepteurs de choisir des formats différents et de définir les intructions du niveau machine. Nous avons remarquer précédemment que instructions et format sont identiques à ceux de l'ordinateur Super-16 conçu par AMD.

Architecture de l'unité centrale

Dans l'unité centrale *(fig. 5)*, tous les transferts de données internes sont effectués par l'intermédiaire du bus interne à 16 bits. Le transfert de données entre le bus système et le bus interne de l'unité centrale est effectué par le registre de données, le registre d'adresse et le verrou Z. Le système présente une structure pipeline à un niveau. A la sortie de la mémoire du microprogramme, le registre agit comme un registre pipeline et peut ainsi exécuter simultanément une microinstruction et la recherche de la suivante. Le registre d'instruction (IR) et le verrou Z permettent une architecture en pipeline au niveau macromachine. Tandis que le décodage de la macroinstruction contenue dans le registre d'instructions s'effectue, le verrou Z peut contenir la macroinstruction suivante (dans un mode registre à registre), ou le champ de déplacement (en adressage indexé), ou les données (en adressage immédiat).

Chemin des données (macroinstruction)

La macroinstruction en provenance de la mémoire centrale est

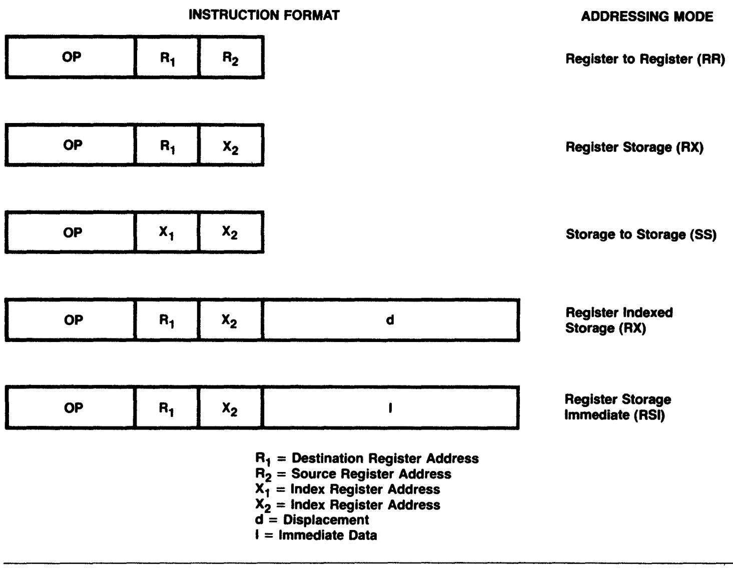
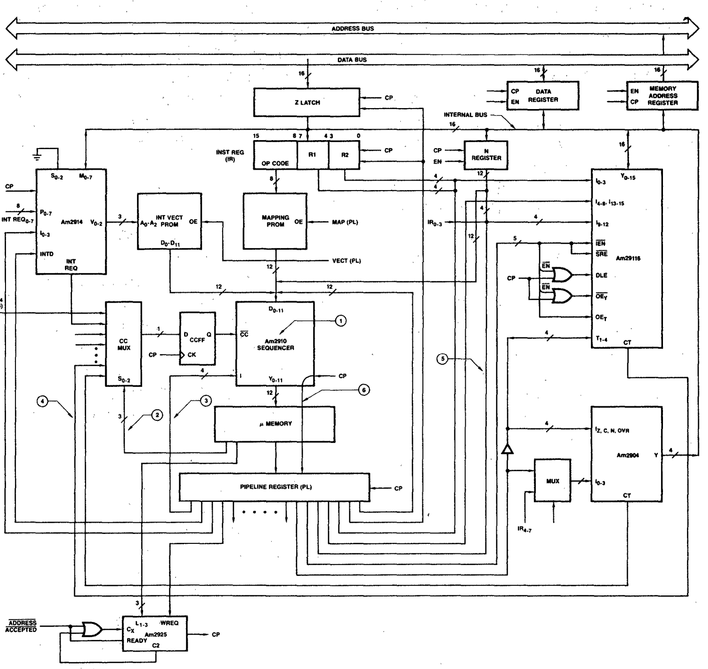

INSTRUCTION FORMAT
ADDRESSING MODE
OP
R_1
R_2
Register to Register (RR)
OP
R_1
X_2
Register Storage (RX)
OP
X_1
X_2
Storage to Storage (SS)
OP
R_1
X_2
d
Register Indexed Storage (RX)
OP
R_1
X_2
I
Register Storage Immediate (RSI)
R_1 = Destination Register Address
R_2 = Source Register Address
X_1 = Index Register Address
X_2 = Index Register Address
d = Displacement
I = Immediate Data

Fig. 4 — Formats des intructions.
— *Instruction formats.*

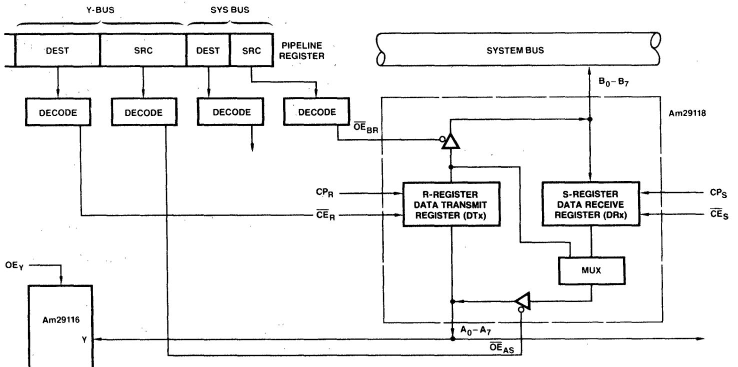
Fig. 5 — Unité Centrale.
— *Central processing unit.*



chargée soit dans le verrou Z, soit dans le registre d'instruction. On peut charger directement une macroinstruction dans le registre d'instruction en rendant le verrou Z transparent, ce qui est fait en forçant son signal de validation à l'état HAUT. C'est le mode d'adressage de la macroinstruction en cours de décodage qui détermine si le chargement se fera dans le verrou Z ou dans le registre d'instruction. Pendant cette opération de remplissage du pipeline, la première instruction est chargée dans le registre d'instruction. Toutes les instructions à mot unique sont ensuite chargées dans le verrou Z puis dans le registre d'instruction après que la macroinstruction en cours ait été exécutée. Puisqu'une instruction à deux mots utilise des informations stockées dans le registre d'instruction (qui contient l'instruction) et dans le verrou Z (qui contient le déplacement), la macroinstruction suivante en provenance de la mémoire centrale sera chargée directement dans le registre d'instruction. Si, au cours du décodage d'une instruction à deux mots, ils s'avère que le contenu du verrou Z une fois décodé est un déplacement, l'Am 29116 s'en servira pendant le cycle de calcul de l'adresse opérande. S'il est décodé comme étant une donnée immédiate, il sera utilisé pendant le cycle d'exécution (voir la partie sur l'exécution des instructions dans ces deux cas).

Le transfert des données entre le processeur et les autres dispositifs utilise des ports d'E/S bidirectionnels Am 29118 à accumulateur *(fig. 6)*. Sous le contrôle du microprogramme, le registre de réception des données (DRX reg) reçoit des données provenant du bus de données du système. Le registre de transmission de données (DTX reg) peut agir soit comme registre de transmission de données sur le bus de données du système, soit comme un registre temporaire pour le bus interne de l'unité centrale. En tant que registre temporaire, et sous le contrôle du microprogramme, on peut charger des données provenant du bus interne ou les lire sur ce bus. Le verrou D du port d'entrée de l'Am 29116 permet de faire entrer et sortir des données au cours du même microcycle. Les données sont amenées par le port Y (à partir du verrou Z ou du registre DTX) et chargées dans le verrou D au cours de la première moitié du cycle. Pendant la seconde moitié du cycle, le verrou D est invalidé. On peut alors valider les émetteurs de sortie pour pouvoir faire apparaître le résultat de l'UAL sur le bus interne.

Le registre N permet d'intégrer des fonctions comme le branchement multivoie et la normalisation. Sous le contrôle d'un masque, il est possible de faire un encodage de priorité sur un mot de 16 bits de l'Am 29116. Le vecteur à 5 bits est chargé dans le registre N et utilisé au cours du cycle suivant soit pour se connecter à un microprogramme spécial, soit comme nombre « n » d'une instruction de rotation d'un mot à l'intérieur du décaleur combinatoire.



$\overline{OE}_{AS}$	$\overline{CE}_R$	Function
0	0	Read R, Disable CP_R
0	1	Read S, Disable CP_R
1	0	Enable CP_R
1	1	Disable CP_R

Fig. 6 — Schéma de l'Am 29118.
— *Am 29118 block diagram.*

Le contrôle par microprogramme

La logique de contrôle génère la séquence adéquate d'exécution du microprogramme. Le contrôleur de microprogramme Am 2910 génère l'adresse de la microinstruction suivante. Il peut effecteur soit un accès séquentiel, soit un branchement conditionnel à toute microinstruction comprise à l'intérieur des 4 Kmots de microinstructions. L'Am 2910 reçoit l'adresse de branchement (aux entrées D) à partir des quatre sources suivantes : registre, pipeline, décodeur du vecteur d'interruption, décodeur de macroinstruction, registre N. Le décodeur de macroinstructions (PROM de conversion) utilise le code opératoire (bits 8 à 15 du RI) comme une adresse, et fournit l'adresse de départ du microprogramme qui exécute chaque macroinstruction. Le décodeur de vecteur d'interruption utilise le vecteur à 3 bits (engendré par le contrôleur à affectation de priorité et à vectorisation Am 2914) unique à chaque élément demandeur et génère l'adresse de traitement de l'interruption. Le registre N permet de faire le branchement multivoie. L'instruction encodage de priorité de l'Am 29116 génère un vecteur binaire à 5 bits qui indique la position de bit de plus forts poids égal à 1 à l'intérieur d'un mot de 16 bits. Ce vecteur ainsi que d'autres bits définis antérieurement peuvent être utilisés comme adresse de branchement.

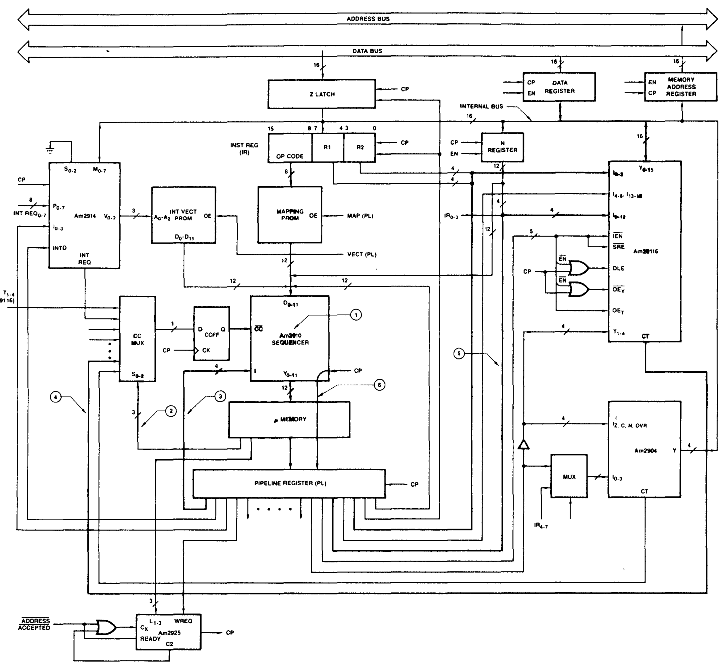
La décision d'effectuer un branchement avec le Am 2910 dépend de la condition présente à l'entrée CC. Les bits d'états provenant du bus-T de l'Am 29116 (retenue, débordement, zéro, négatif) peuvent être utilisés pour faire un branchement au cours du cycle suivant en sélectionnant une entrée sur quatre (T_1 à T_4) au MUX-CC. La sortie de test conditionnel de l'Am 29116 ou Am 2904 peut être sélectionnée sur le MUX-CC afin de pouvoir faire un branchement à partir de conditions complexes (comme inférieur à, égal à,...) ou des conditions engendrées intérieurement. La *figure 7* décrit l'analyse avec diagramme de temps du microcycle pour quelques chemins représentatifs. Cette analyse est effectuée en utilisant le temps de propagation le plus défavorable de chaque circuit. Le temps de propagation le plus important affectant le microcycle est celui qui passe par le registre pipeline (de CP à Q), l'Am 29116 (de I à CT), le MUX-CC (de D à Y), l'Am 2910 (de CC à Y), la mémoire du microprogramme (temps d'accès par rapport aux adresses) et retourne au registre pipeline (temps de pré-positionnement).

Cependant, l'utilisation de la bascule D (CCFF) sépare le temps de cyle en deux temps non critiques. Puisque la bascule retarde le signal du code de condition (CC) à l'entrée de l'Am 2910 d'une impulsion d'horloge, les lignes de sélection du MUX-CC sortent de la mémoire du microprogramme pour exécuter en parallèle la sélection du code condition et la microinstruction. Quand le CCFF est actif, le temps le plus critique du système passe par l'Am 2910 (de CP à Y), la mémoire du microprogramme (temps d'accès par rapport aux adresses) et le registre pipeline (temps de pré-positionnement). Ce chemin peut être rendu non critique par l'adoption d'une architecture à deux niveaux : mais pour une raison de simplicité, il n'en sera pas fait mention ici.

Exécution de l'instruction

Le cycle d'instruction normal comprend quatre séquences d'opérations : formation de l'adresse mémoire de l'instruction, recherche de l'instruction, décodage de l'instruction, exécution.

Pour les instructions qui nécessi-



TIMING ANALYSIS

Path 1				
	Instruction Register	CP	Q	12ns
	Mapping PROM	ADD	Y	35ns
	Sequencer	D	Y	20ns
	Micro-memory	Add	Y	40ns
	Pipeline Register	Set up		5ns
				112ns
Path 2				
	CC Flip Flop	CP	Q	12ns
	Sequencer	CC	Y	43ns
	Micro-memory	Add	Y	40ns
	CC-MUX	Sel	Y	15ns
	CC Flip Flop	Set up		5ns
				115ns
Path 3				
	Pipeline Register	CP	Y	12ns
	Sequencer	I	Y	70ns
	Micro-memory	Add	Y	40ns
	Pipeline Register	Set up		5ns
				127ns
Path 4				
	Pipeline Register	CP	Y	12ns
	Tri-State Gate	Enable	Y	29ns
	Am29116	I	CT	48ns
	CC-MUX	DIN	Y	15ns
	CC Flip Flop	Set up		5ns
				109ns
Path 5				
	Pipeline Register	CP	Y	12ns
	Tri-State Gate	Enable	Y	29ns
	Am29116	I	Y	79ns
	Data Reg	Set up		5ns
				125ns
Path 6				
	Am2910 (PC and Stack)	CP	Y	100ns*
	Micro-memory	Add	Y	40ns
	Pipeline Register	Set up		5ns
				145ns

*It is assumed that the previous instruction could produce no change in the counter or could only decrement the counter.

Fig. 7 — Analyse temporelle des séquences.
— *System timing analysis.*

tent des opérandes provenant plutôt de la mémoire que du fichier-registre local, deux étapes supplémentaires devront être menées après l'opération de décodage : formation de l'adresse de l'opérande et recherche de l'opérande. Les performances sont améliorées si on fait une recherche préliminaire de l'instruction suivante au cours de l'exécution de l'instruction. L'architecture en pipeline n'utilise que partiellement cette capacité de superposition des opérations de recherche, de décodage, d'exécution des différentes instructions car l'Am 29116 agit en même temps comme « Unité de Commande de Programme » et comme « Unité Arithmétique Logique ». Si on utilise d'autre UCP séparément (2901 ou 2930), on augmentera les performances du système et on utilisera au maximum les possibilités de l'architecture en pipeline.

Formation de l'adresse mémoire (un microcycle)

Le compteur ordinal (PC) est incrémenté de 2 et le résultat est chargé dans le MAR (registre d'adresse mémoire) et retourne au PC. Cela peut être effectué en sélectionnant la RAM comme destination et en mettant OE_Y à l'état BAS dans l'Am 29116.

Recherche de l'instruction (un microcycle)

Les signaux demande de mémoire principale et échantillonnage de données sont générés. La mémoire utilise l'adresse au cours du même cycle et place les données sur le bus. L'instruction est chargée dans le registre d'instruction au prochain front de montée du microcycle.

Décodage de l'instruction (un microcycle)

L'instruction est chargée dans le registre d'instruction en passant par

L'Am 29116

L'Am 29116 permet au processeur de réaliser des fonctions logiques et arithmétiques puissantes. En outre, l'Am 29116 conserve et génère les adresses de la mémoire centrale. Sa RAM à 32 mots fournit un espace de stockage temporaire qui améliore le débit du processeur. L'utilisateur peut se servir de la moitié inférieure du fichier à 32 registres (reg. 0 à 15) comme « brouillon ». Le système d'exploitation utilise la moitié supérieure (reg. 16 à 31) pour conserver la trace des piles mémoire, du compteur, etc. Pour implanter un langage évolué, le décaleur combinatoire permet une insertion et une extraction de champ, une rotation et une recherche en table. Un registre d'état incorporé permet de tester les bits définis par l'utilisateur et les autres bits d'état par l'intermédiaire soit des entrées d'instructions, soit du bus-T bidirectionnel, le résultat apparaissant à la sortie CT.

Le jeu d'instructions de l'Am 29116 fournit au processeur une puissance supplémentaire. Les instructions de priorité et de rotation peuvent être utilisées pour normaliser une instruction à virgule flottante. Sa capacité de manipulation de bits facilite la manipulation d'adresses et la génération de données.

Le diagramme détaillé de l'unité centrale *(fig. 5)* indique que les 16 entrées « instruction » de l'Am 29116 sont commandées par les bits du microcode. Sur les bits d'entrée 0 à 3, le multiplexeur (bus à trois états) permet un adressage des 16 registres, soit par le champ de macroinstruction (R1 ou R2), soit à partir du microcode. De même, pour contrôler le nombre « n » (utilisé pour effectuer la rotation d'un mot) à partir de la macroinstruction, du registre N ou de la microinstruction, on utilise un multiplexeur (bus à trois états) à l'entrée des bits 9 à 12. Les sorties des champs R1 et R2 du registre d'instruction sont commandées par des bits différents du microcode. Comme l'Am 29116 possède une RAM à port unique, deux cycles sont nécessaires pour réaliser une opération de registre à registre. Pendant le premier cycle, le contenu de R2 est transféré vers l'accumulateur. Pendant le deuxième cycle, une opération entre R1 et l'accumulateur est réalisée, R1 constituant la destination. Quatre bits supplémentaires du microcode commandent les entrées T1 à T4 de l'Am 29116. Cela permet d'exécuter simultanément le test d'états et une instruction. Les 4 bits d'états (C, N, OVR, Z) peuvent être chargés dans le registre d'état ou sortis par le bus-T, qui permet également un chargement sélectif du bit d'état à l'intérieur de l'Am 2904.

L'Am 2904 réduit le nombre de cycles nécessaires à la réalisation de manipulations d'états. Pour pouvoir se brancher au cycle suivant à partir de ces bits d'états (C, N, OVR, Z), le bus-T est envoyé sur la MUX-CC.

L'Am 2904 possède deux registres d'états, l'un correspondant à l'état au niveau micromachine, et l'autre à l'état au niveau macromachine. Le micro-état est remis à jour à chaque microcycle au cas où le port T de l'Am 29116 est le port de sortie. Le macro-état est remis à jour à la fin de chaque instruction-machine. Ainsi l'Am 2904 facilite le branchement au niveau micro ou macromachine.

L'émetteur et générateur d'horloge du système, l'Am 2925, assure la génération de l'horloge principale et de quatre signaux d'horloge de forme différente (rapport cyclique différent). L'Am 2925 a la possibilité de modifier dynamiquement la longueur du microcycle, et il réagit à un interface asynchrone par la génération d'un état d'attente. Ces deux possibilités améliorent la puissance de l'UC en facilitant l'interface avec des dispositifs plus lents.

le verrou Z. La PROM de conversion génère une adresse de départ appropriée pour le microprogramme à partir du code opératoire.

Formation de l'adresse de l'opérande (un microcycle)

Après chaque recherche d'instruction, il se produit un autre cycle de lecture. Les données sont stockées dans le verrou Z. Selon le mode d'adressage de l'instruction pendant le cycle de décodage, la logique de décodage détermine si les données se trouvant dans le verrou Z composent l'instruction suivante, le déplacement ou les données immédiates. La valeur de déplacement et le registre index spécifique sont utilisés afin de former une adresse d'opérande qui sera chargée dans le MAR.

Recherche de l'opérande (un microcycle)

L'opérande est lu à partir de l'adresse mémoire générée au cours de la précédente période d'horloge et stockée dans le verrou Z (ou reg. D).

Exécution

L'Am 29116 réalise l'opération spécifique sur les opérandes. Deux microcycles sont nécessaires pour une instruction de type registre à registre car le fichier registre de l'Am 29116 ne possède qu'un port. Cette opération est réalisée en un cycle par des architectures à deux adresses. Dans une instruction de type registre à registre (RR), une deuxième recherche en mémoire suivant le cycle de recherche de l'instruction se produit pendant l'opération de remplissage du pipeline. La seconde instruction recherchée est stockée dans le verrou Z. L'instruction suivante est chargée du verrou Z dans le R1 après que l'instruction en cours ait été exécutée. Simultanément, l'adresse de l'instruction suivante à rechercher est placée sur le bus d'adresse. Pendant le cycle suivant, l'instruction tirée de la mémoire est stockée dans le verrou Z. L'exécution de chaque instruction prend deux périodes d'horloge, comme nous l'avons déjà expliqué.

Tableau I

Champ des microinstructions.
Microinstruction fields.

Field Width	Mnemonic	Description
ALU		
16	I_0-I_{15}	29116 Instruction
1	DLE	29116 Data Latch Enable
1	IEN	29116 Instruction Enable
1	OEY	29116 Output Enable Y-bus
1	SRE	29116 Status Register Enable
1	OET	29116 Output Enable T-bus
3	RAMSRC	29116 I_0-I_4 Source Select
2	NSRC	29116 I_9-I_{12} Source Select
Data Path		
4	DSEL	Data Register Source/Destination Select
1	DLD	Data Register Enable
1	MARLD	Memory Address Enable
1	IRLD	Instruction Register Enable
1	ZLD	Z Latch Enable
1	NLD	N Register Enable
1	MAP	Mapping PROM Output Enable
1	VECT	Interrupt Vector PROM Output Enable
Memory Control		
1	R/W	Memory READ/WRITE Pulse
1	WREQ	Wait Request
1	DATASTB	Data Strobe
1	MEMREQ	Memory Request
Interrupt Control		
4	I_0-I_3	2914 Instruction
1	INTD	2914 Interrupt Disable
Clock Select		
3	L_1-L_3	2925 Clock Length Select
Status		
1	EZ	2904 Enable Zero
1	EC	2904 Enable Carry
1	ES	2904 Enable Sign
1	EOVR	2904 Enable Overflow
1	CEM	2904 Enable Machine Status
1	CEMICRO	2904 Enable Micro Status
Test		
4	T_1-T_4	29116 or 2904 Test Status Instruction
CCSEL		
3	CCSEL	Condition Code MUX Select
Sequence Control		
4	I_0-I_3	2910 Instruction
Branch Address		
12	BA	Next Micro Address
78		

Sauf pour l'opération de remplissage du pipeline, l'Am 29116 est utilisé dans tous les cycles, soit pour l'exécution d'une instruction, soit pour la génération de l'adresse de la mémoire centrale.

Pour un stockage de type registre à index (RX), au cours de l'opération de remplissage du pipeline, le déplacement est recherché immédiatement après l'instruction. Une fois que le pipeline est rempli, la génération de l'adresse de l'instruction suivante et la recherche d'un opérande peuvent être effectuées simultanément. L'Am 29116 est utilisé dans six cycles

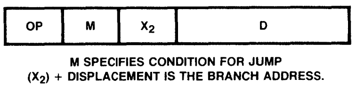

OP
M
X2
D
M SPECIFIES CONDITION FOR JUMP
(X2) + DISPLACEMENT IS THE BRANCH ADDRESS.

Fig. 8 — Instruction de branchement conditionnel.
— *Branch on condition (RX) instruction format.*

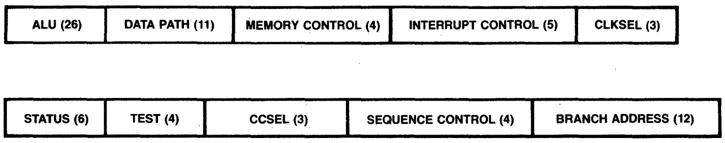


Fig. 9 — Format d'une microinstruction.
— *Microword format.*

sur les septs nécessaires à l'exécution de l'instruction de type RX.

La décision de se brancher sur l'instruction du type RX s'effectue après que l'unité centrale ait déterminé si la condition était vraie ou fausse (pendant le cycle d'exécution A). La *figure 8* décrit le format du macrobranchement. Le code de condition pour le branchement est présenté à l'Am 2904, qui le présente à son tour à l'Am 2910 où sera prise la décision.

La microinstruction possède 78 bits *(fig. 9).* Les bits de commande de chaque unité fonctionnelle sont regroupés pour faciliter la lecture.

Le *tableau I* montre les définitions de chaque bit de commande. Les signaux de validation du chargement des verrous Z et D de l'Am 29116 peuvent être utilisés avec les horloges ou d'autres formes de signaux de synchronisation pour permettre un fonctionnement correct.

Les signaux d'interface mémoire peuvent être générés à partir des signaux d'échantillonnage de données et de lecture/écriture, selon le type de RAM statiques utilisées dans la mémoire centrale.

Conclusion

Les possibilités de microprogrammation des dispositifs en tranches de bits rendent la conception d'une architecture personnalisée plus facile et plus rapide.

La réalisation décrite dans cet article a été effectuée avec un nombre minimum de composants. Le débit du système peut être augmenté en ajoutant une unité de commande de programme séparée qui, en travaillant en même temps que l'Am 29116 permet de libérer ce dernier de la génération d'adresse de mémoire centrale.

RECOMMANDATIONS AUX AUTEURS

« Electronique, Techniques et Industries » paraît en dix fascicules illustrés de figures dans le texte (dessins ou photographies). Il sera demandé aux Auteurs une participation pour les articles dépassant six pages imprimées.

Les articles seront fournis en deux exemplaires (texte tapé en double interligne, résumé français et anglais, références, figures, légendes en français et en anglais, et courte bibliographie).

Sous le titre, les auteurs indiqueront leurs prénoms et noms, ainsi qu'une désignation succincte, mais complète, de leur organisme - notamment l'adresse.

Les articles doivent obligatoirement être accompagnés d'un résumé de 200 mots environ, précisant clairement les points essentiels développés dans l'article. L'auteur devra fournir la traduction en anglais du titre de l'article et du résumé.

Les références seront numérotées continûment, renvoyant à la liste bibliographique indiquant le nom de l'auteur accompagné entre parenthèses de l'initiale de son prénom, le titre de la publication, le titre de la revue, l'année, la tomaison ou le cas échéant le numéro, les pages de début et de fin d'articles et, pour le cas d'un livre, l'éditeur, le lieu et l'année d'édition.

Les figures au trait seront présentées sur calque ou sur papier blanc, repérées au crayon par leur numéro d'ordre correspondant à la feuille de légendes. Ne pas oublier qu'une lettre mal tracée, trop petite ou trop grasse supporte mal la réduction ; se rendre compte de ce que donnera, à l'échelle, le schéma inséré dans une colonne de la Revue, la hauteur des lettres étant au moins de 1,5 mm.

Les photographies, positifs contrastés, sur papier brillant, seront à l'échelle suffisante pour ne pas perdre trop de détails lors de la réalisation des clichés d'imprimerie. Reporter le nom de l'auteur et l'abrégé du titre au dos de chaque photographie, calque ou feuille de figures.

Dans le cas de reproduction partielle ou totale de texte ou d'illustration, l'Auteur en avisera le Rédacteur en Chef afin que l'Editeur obtienne, s'il y a lieu, les autorisations nécessaires.

On écrira les formules et équations avec le plus grand soin, à la main toutes les fois que la typographie machine entraînerait une ambiguïté de lecture.

Les articles doivent être adressés au Rédacteur en Chef d'Electronique, Techniques et Industries, 12, rue de Seine - 75006 - PARIS (Tél. : (1) 325.58.74).

A new concept in backplane communication

Eric Chou and *Gary Connor* of AMD describe a semiconductor device that can be used to minimise problems of RFI, EMI and noise immunity in backplane wiring.

Modern computer designs are moving in the direction of more compact circuitry with increasing levels of power dissipation. This in turn has lead to a rising occurrence of noise generation in the form of RFI (radio frequency interference) and EMI (electromagnetic interference). FCC regulations limit such noise and enforcement has been stepped up. The need to restrict such noise occurrences has become more acute since FCC article 15 went into full effect on October 1, 1983. Violators of these regulations may now find their merchandise blocked from sale to the market until compliance is achieved.

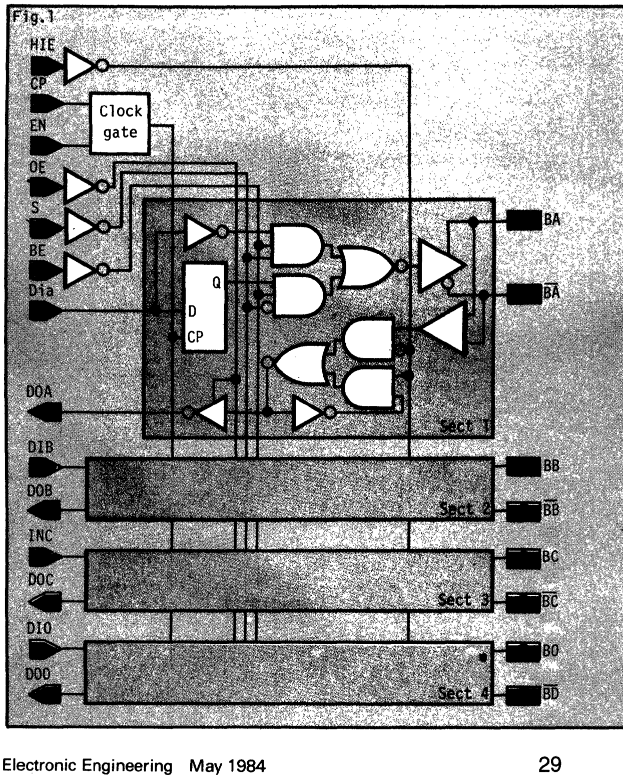
While RFI and EMI noise can occur in many areas of a system the principal source is the backplane or board-to-board interconnection. These interconnections often take place via long parallel wires or PC cards with strips of copper spaced to form a cable. Without caution these act as antennae and provide a source of unwanted radiation and adjacent signal coupling (noise). There are several ways to limit the problems and the two prime choices are shielding and twisted pair cabling. The most effective method involves the use of twisted pair or an approximation to twisted pair cable driven-and-received differentially. This can be enhanced by adding shielding to encompass the cable assembly.

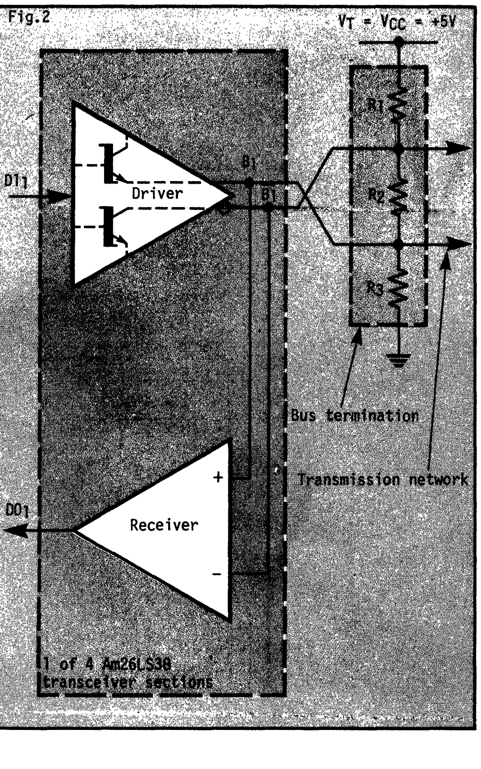
Differential signalling provides two forms of helpful cancellation. First, radiated signal caused by the dv/dt in each signal lead is opposed by its equal and opposite compliment, thus cancelling the RFI. Second, terminating the cable pair line to line in its characteristic impedance assures that current flowing in one signal lead is opposed equally by its mated pair, thus minimising EMI. All of these techniques require a driver-receiver pair capable of operating in a balanced differential mode without distortion.

A transceiver capable of such operation is the Am26LS38 quad differential backplane transceiver. Designed to integrate Schottky TTL performance, high noise immunity and wired logic capability into a low cost differential backplane structure it provides users with a means of designing an error free backplane. By utilising this device a backplane with multiple transceivers in a party-line, wired-OR logic configuration can be achieved. These IC's operate from a single 5V power supply with such features as output disable during power-up and down, guaranteed fail-safe, thus assuring reliable systems operation.

Figure 1: Black diagram of the AM26LS38 quad differential backplane transceiver

Figure 2: A typical system connection diagram





The Am26LS38 has been designed specifically for applications with differentially driven wire, trace or plane transmission systems of controlled impedance.

These transmission mediums have proven to be useful in preventing unwanted signalling.

Cable losses

The receiver is designed with a low (±50mV) threshold that combines with its driver output of greater than 500mV to provide systems attenuation tolerance and margin for cable losses. A 25mV (typical) hysteresis minimises the switching sensitivity caused by lines with slow transition times.

The devices will operate in the presence of up to 1.5V of common mode voltage thus eliminating the need for absolute ground references and has symmetrically controlled rise/fall times reducing the radiated harmonic disturbance.

A block diagram of the Am26LS38 is given in figure 1 and backplanes designed using it have three main elements: a driver section; a receiver section; and a controlled impedance differential line with pre-biasing line termination. Figure 2 shows a typical system connection. The device has two operational states; active (driver outputs on) and passive (off).

The 2-state driver employs an active pull-down (open collector) and active pull-up (open emitter) output stage which are simultaneously applied to a transmission line to form the driven signal (figure 3).

When a driver is active, both output stages turn on impressing a minimum voltage differential of 0.5V on the bus. This driver reverses the voltage across R_2 and causes the receiver to detect a logic '1'. In the passive mode both output stages are off and voltage levels, observing polarities, return to the conditions set by the bias network thus maintaining a logic 'O' at the receiver input.

In either state the voltage across the differential lines is symmetrical about Vcc/2 (dependent on R_1 and R_3). Since the voltage levels required to change state are close together, high bus data rates (up to 10MHz) can be achieved.

Figure 3: The scheme for driver, receiver and termination resistors

Figure 4: Use of the AM26LS38 within a processor complex

Users may also use the two state (active/passive) operation for wired logic functions.

With the lines assuming a known passive state, a false (wired-OR) or a true (wired-AND) can be accomplished providing users with a means for polling and prioritising transmissions.

The Am26LS38's shared-bus digital balanced line transmission is useful for the inter-element backplane communication within a processor complex. An example of this application area is given in figure 4. In computer interconnection both data and address buses connect the processor, memory and peripherals via backplane networks.

Typical network

For such an application the bus line is bi-directional and party lined, but the subsystem internal buses are uni-directional. While network dimensions vary with the types and size of cabling utilised, a typical network of up to 25 feet can be created with data rates in excess of 10MHz.

The Am26LS38 has the capacity to drive a terminated twisted pair, ribbon or strip cable and multiple transceivers with proper design.

Fig.3

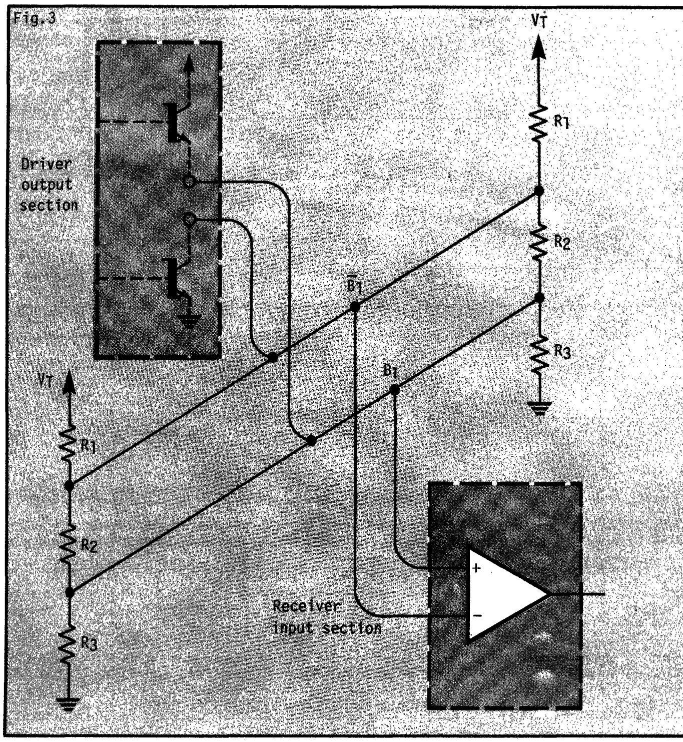
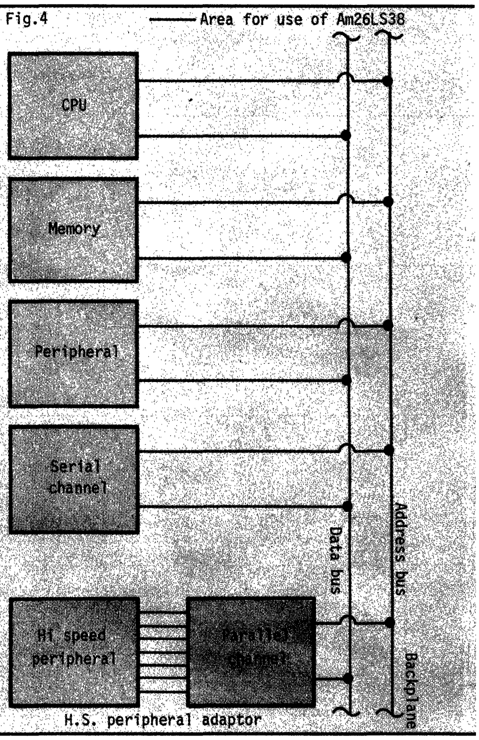


Fig.4



The data path for one of the bus interface is shown in figure 5. Two modes of bus operation are possible, these are: registered transmit and latched receiver of buffered transmit and latched receiver.

With the register and latch combination a storage mode can be established for the output and input data ('register mode') for direct bus applications. By incorporating storage on-chip, improved speed and a lower package count is achieved.

Discernible signals

It has been found that with the use of symmetric termination a reduction of common-mode reflections in the bus line can be achieved and this makes individual signals more discernible. Performance can be further improved upon by centering the network at $V_{CC}/2$ ($R_1=R_3$).

This allows induced noise and reflections to appear as a common mode signal at the receiver. Since the AM26LS38 can operate with up to 1.5V of common mode voltage the device can maintain a 0.5V minimum voltage differential on the bus.

A first order approximation of termination resistor values may be developed by letting the ratio of R_1 to R_2 be 2:1 and the Thevenin equivalent resistance of the termination equal the characteristic impedance of the line (Zo — see figure 6).

Assuming $R_1=R_3$

$$V_{OC} = V_T (R_2/(2R_1+R_2)) \quad 1$$

$$R_{TH} = 2R_1 R_2/(2R_1+R_2) \quad 2$$

from equation (1) and (2):

$$R_1 = V_T R_{TH}/(2V_{oc}) \quad 3$$

$$R_2 = V_T R_{TH}/(V_T - V_{oc}) \quad 4$$

If $V_T = 5V$,

$V_{OC} = 1V$,

and $R_{TH} = 90\Omega = Zo$

we can derive that

$R_1 = 225\Omega$,

$R_2 = 112.5\Omega$.

Second order adjustments require attention to unit loading factors (receiver differential input resistance is in parallel with R_2) transmission rates and a host of other factors.

Reduction of noise and tolerance to distrubances that occur are a major concern of backplane designers seeking to create an error free system communication. Attentuation, reflection effects in cables, differential noise, mutual inductance, differences in gain for ac and dc voltages and common mode voltage (CMV) are among the problems facing these designers.

Low inductance wiring, ground planes on circuit cards, and twisted-pair cable all provide low values of coupling and controlled impedance that aid in the prevention of signal degradation.

Removing errors

Similarily, coupling effects can be lowered by bringing the common ground closer to both lines.

Shielding, a proper ground systems and paired line configurations are also effective for limiting crosstalk due to current cancellation and differential voltage cancellations. Measures like these when used with the Am26LS38 will allow designers to create error free backplane systems.

The Am26LS38 has been designed to solve the backplane interconnection problem without systems compromise and the cost that of differential transmission, a small price to pay for noise free, error free signalling.□

Figure 5: The data path for one bit of the bus interface

Figure 6: Circuit used for calculating first order approximation of termination resistor values.

Fig.5

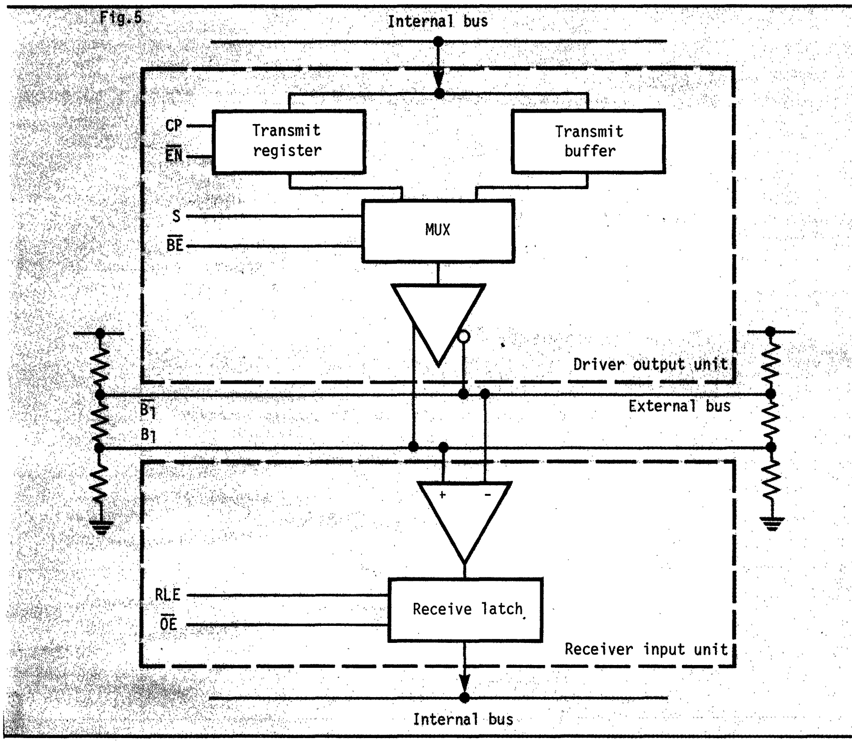
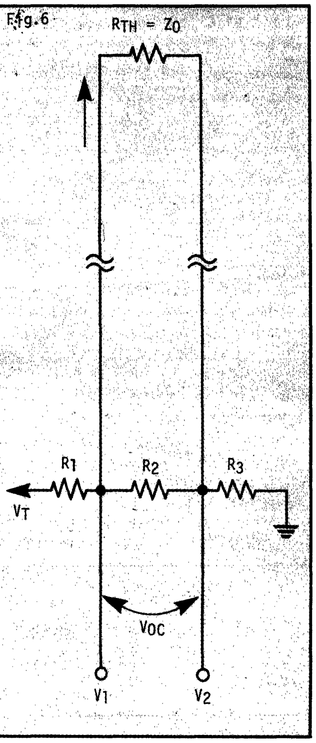


Fig.6



PUTTING THE SPEED INTO 12-BIT A-TO-D CONVERSION

Illustration by Richard Evans

by Edgar R. Macachor,
Advanced Micro Devices

Until recently, users of analogue-to-digital converters (ADCs) have had to choose between costly, multi-chip hybrids to achieve high speeds or single chip converters with much slower operating times. Fast ADCs having conversion times of <5μs are structured as hybrids in a multi-chip form. The conversion technique used, especially with 12-bit ADCs achieving fast conversion times, may also be of a hybrid type combining successive approximation and parallel conversion technique, or straight successive approximation. However, hybrid ADCs are expensive, and they have not been designed to interface directly with either 8-bit or 16-bit microprocessors.

Monolithic, microprocessor-compatible, 12-bit ADCs that are available utilise CMOS or I^2L technology, for low device power dissipation, and are of the integrating or the successive approximation types. Their conversion times are too slow at 25μs or more and often require software and/or hardware overhead just to check the status of the ADC.

The Am6112, however, is a monolithic 12-bit ADC with a fully microprocessor-compatible architecture designed to operate at 3μs. The device uses successive approximation to provide higher performance with improved density due to a smaller logic cell (Fig. 1). The DAC portion is untrimmed since it is inherently monotonic and linear by means of error averaging. Internally, the control logic is translated from the TTL-compatible input control signals to linear-differential logic (LDL), a non-saturating form of logic similar to ECL.

However, because the LDL speed-power product is significantly lower than that of ECL, the result is that the device can convert an analogue signal in 3μs, with 12 bits of accuracy, in any one of the five operating modes selected by the user. The open-collector $\overline{ACK}$ line of the converter allows it to be wired-OR with other I/O devices.

Depending upon the throughput rate required, the user can select one of five operating modes. For example, mode 0 is suitable for microprocessor-based systems with low throughput rate requirements. Mode 2 handles fast data transfer rates and is convenient because data outputting is done automatically. For systems requiring medium throughput rate, the Am6112 operates mode 1.

The ADC is initialised by writing into the command register via the bidirectional data pins D2-D0. The mode of operation and the digital output code desired are determined by the data bits D2-D0 except in mode 4 (stand-alone mode) where the digital output code defaults to offset binary.

After initialisation, a conversion cycle is started by the positive transition of a $\overline{WR}$ pulse which is generated by an I/O write instruction. Valid data may be read out as 4MSBs and 8LBs or as 8LSBs first followed by the 4MSBs, depending on the status of C/$\overline{\mathrm{D}}$ input pin during an I/O read instruction. This means that the valid 4MSBs may be read out although the device hasn't completed the full 12-bit conversion process.

Mode 0 operation required three I/O instructions after initialisation. An I/O write starts a conversion cycle and two I/O read commands read the full 12-bit valid data out in two bytes. During a data read, the ADC's internal status line ($\overline{\mathrm{CC}}$) is sampled by $\overline{\mathrm{CS}}$ and $\overline{\mathrm{RD}}$. In modes 0, 1, and 2 the $\overline{\mathrm{ACK}}$ reflects the status of the ADC during an active $\overline{\mathrm{RD}}$ period. For example, when $\overline{\mathrm{CC}}$ is high during the entire $\overline{\mathrm{RD}}$ period, the ACK will also be high. The $\overline{\mathrm{ACK}}$ line can then be used to extend the I/O read cycle if required. This condition occurs when the 4MSBs are read even though they are not yet valid. In this case, the ACK is then used to insert wait states and extend the microprocessor I/O read cycle until the full 12-bit conversion is done.

In mode 1, a conversion cycle is started by the positive transition of a $\overline{\mathrm{RD}}$ pulse generated by an I/O read instruction (Fig. 2). The reading of the 12-bit data is then similar to mode 0. Mode 1 lends itself well to microprocessors such as the Z80 and the Z8000, having I/O data block transfer capability as part of their instruction repertoire. The (INIR) or (INDR) instruction of the Z80 and the (INIRB) or (INDRB) instructions of the Z8000 are capable of starting conversion cycles and then reading the 12-bit data from the Am6112 until the desired amount of data is stored in memory.

The first 12-bit data is invalid data since the first conversion cycle is not started until the positive transition of the second RD pulse. This is not a disadvantage because in some applications, such as waveform analysis, the first sample data may be discarded. This is due to the necessity of oversampling the analogue waveform. The $\overline{\mathrm{ACK}}$ line has the same characteristics as that of mode 0.

Mode 2 puts the ADC under the control of a direct memory access (DMA) controller. DMA is a means of transferring data between peripheral devices and system memory at a much faster rate compared with the CPU-controlled data transfer modes 0 and 1. During DMA transfer, the CPU is basically disabled and the controller provides all the signals to control the process.

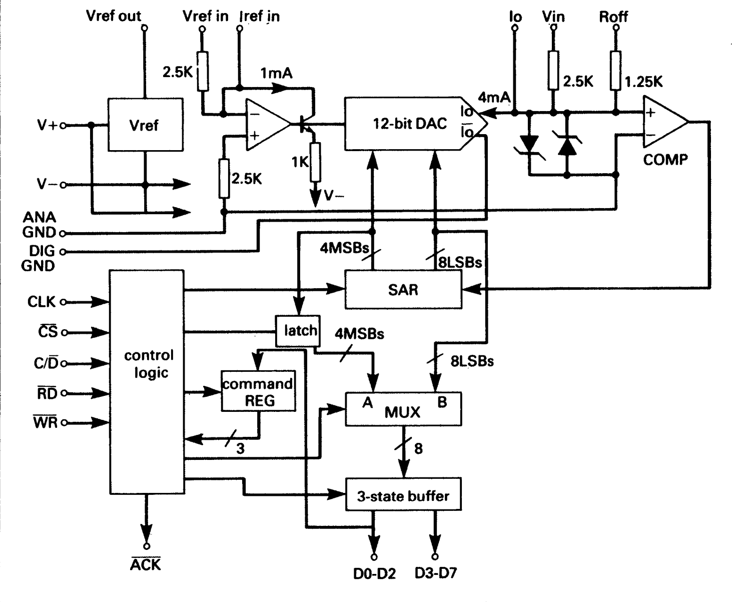


Fig. 1: Am6112 simplified block diagram.

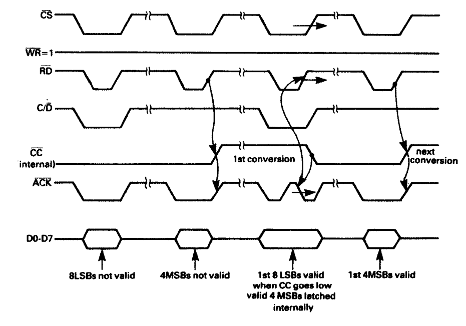
In mode 2, the ADC internally controls the output of the data bytes. The 8LSBs are read out first and simultaneously the 4MSBs are latched internally. The next read cycle outputs the 4MSBs and at the same time starts another conversion cycle to optimise the throughput rate (Fig. 3).

Mode 3 is similar to mode 0 except that the $\overline{\mathrm{ACK}}$ line reflects the true status of the ADC. The difference in the decoding of the $\overline{\mathrm{ACK}}$ line in mode 0 and mode 3 allows the user flexibility in the handshaking with the microprocessor. In mode 3, tying the inverted $\overline{\mathrm{ACK}}$ line to the microprocessor ready or wait input pin may cause additional and unnecessary wait states and will reduce the throughput rate. However, this does guarantee full 12-bit conversion cycles. In mode 0, the number of wait states are minimised because they are inserted only during the active I/O cycle.

Wait states are added when the set-up time requirement at the microprocessor ready input is satisfied. If this set-up time is not met, then there may not be enough wait states inserted to accommodate the ADC conversion cycle. In this case, the converter is "short cycled" and will possibly result in invalid data. The outputting of the data bytes is controlled externally via the C/$\overline{\mathrm{D}}$ line as in mode 0 and mode 1.

The simplest way to operate the Am6112 is the stand-alone mode (Fig. 4). When the ±5V supply is

Fig. 2: Mode 1 timing.


$\overline{\mathrm{CS}}$
$\overline{\mathrm{WR}}$=1
$\overline{\mathrm{RD}}$
C/$\overline{\mathrm{D}}$
$\overline{\mathrm{CC}}$ internal)
1st conversion
next conversion
$\overline{\mathrm{ACK}}$
D0-D7
8LSBs not valid
4MSBs not valid
1st 8 LSBs valid when CC goes low valid 4 MSBs latched internally
1st 4MSBs valid

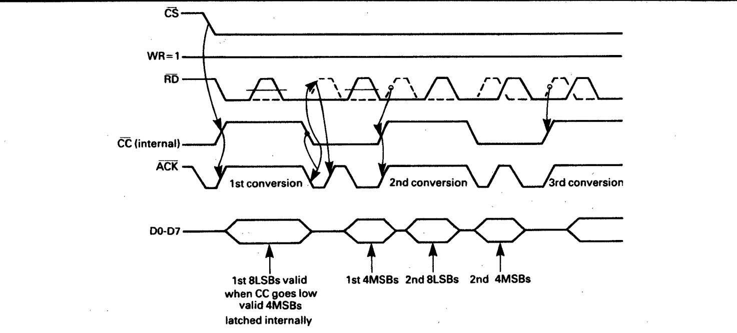


Fig. 3: Mode 2 timing.

first turned on, the device is reset internally to operate with $\overline{CS}$ grounded at all times. Conversion cycles are started at the positive transition of the $\overline{START}$ pulses which are applied to the $\overline{WR}$ input pin. The $\overline{START}$ pulses, which can be as narrow as 100ns, are inverted and applied to the $\overline{RD}$ input pin so that data can be read immediately after a conversion cycle is started. In this mode, the digital output code is expressed only as offset binary.

The single, open-collector, status output line ($\overline{ACK}$) is tied back to the C/$\overline{D}$ input pin to provide a simple means of steering the 12-bit data out in two bytes. When a $\overline{START}$ pulse is applied, the $\overline{ACK}$ line goes high to indicate a conversion cycle. Since the $\overline{ACK}$ line is connected to the C/$\overline{D}$ line, the 4MSB are set up to be read out first. Then, the 4MSB may be latched at the end of the conversion cycle. $\overline{ACK}$ goes low, which also sets up the 8LSBs to be read out. The 8LSBs are then latched between the time $\overline{ACK}$ has gone low and the negative transition of the next $\overline{START}$ pulse.

If the ADC is fed with a clock rate of 4MHz and $\overline{START}$ pulsewidths of 100ns, a throughput rate of 250kHz is easily achievable since the conversion time is only 3.3μs.

The device's 8-bit output data structure enables it to be easily designed in systems using 8-bit CPUs. In minimum system configuration with DMA capability, the 8085 CPU allows for the programming of the Am6112 to operate in modes 0, 1, 2 and 3 (Fig. 5). Except for mode 2, the CPU controls the conversion cycles and the transfer of data which can be stored in the CPU's internal register for immediate processing or in system memory for the case of data collection.

The ACK output of the ADC is fed into the control logic block to control the ready input of the 8085 and the DMA controller. This is done to insert wait states when the ADC conversion cycle is longer than the I/O instruction cycle of the CPU, and this condition is dependent upon the clock rates of the 8085 and the converter. The $\overline{ACK}$ signal also controls the Am6420 sample (track) and hold (S/H) so that while it is logic high the S/H is in the hold mode, and when it goes low the S/H is set to the tracking or sampling mode.

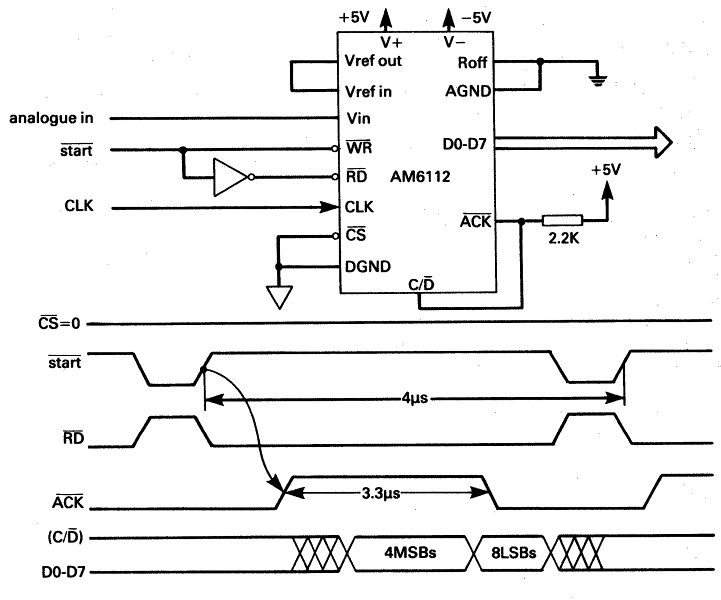
DMA transfer is requested on channel 0 (Fig. 5). The DMA controller responds by setting the HREQ line high to tell the 8085 that it wants to control the bus. The CPU relinquishes its control of the bus by making the hold acknowledge (HLDA) active. The CPU is then basically disabled, and the controller provides all the signals necessary to control the DMA transfer process.

The DMA controller has four channels (DREQ0-DREQ3) where four peripheral devices may request service. DMA request may be hardware or software generated. Each channel has four 16-bit registers (base and current address, and base and current word count) and a mode register accessible via address lines A0-A3. The base registers allow the channels to be automatically re-initialised at the end of a DMA transfer. Auto-initialisation is selected by programming the mode register. Other internal registers are programmed to set the desired operations and options.

Each channel has four modes of operation selected via the mode register. Two of these are the single transfer mode and the block transfer mode. In the single transfer mode, a word is transferred for each DMA request. Block transfer allows the controller to make continuous transfer until the word count for the active channel goes to zero.

The minimum 8-bit system with DMA capability described is adequate in some applications where parallel processing is not really necessary. The inadequacy stems from the

Fig. 4: Stand-alone operation.



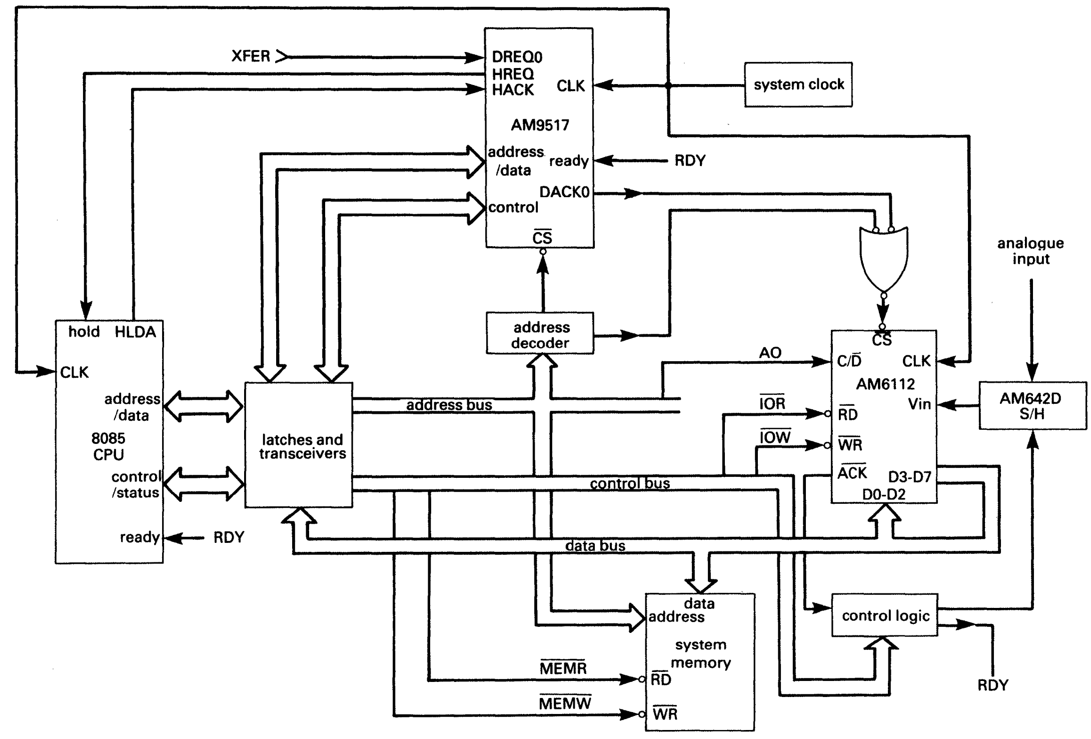


Fig. 5: Am6112 in minimum 8-bit system configuration with DMA capability.

shared-bus architecture. Because the DMA controller cannot concurrently control the buses with the 8085 CPU, the CPU is idle while DMA transfer is in progress. For large data block transfer, this period can be very significant even though the transfer per byte of data is extremely fast.

The limitation associated with the shared-bus structure can be solved by dedicating a device, such as another CPU, to deal with the ADC and other I/O devices in conjunction with the main CPU. However, most general-purpose CPUs are not really suited for the task of I/O device control and processing and will probably still need a DMA controller to provide DMA compatibility.

A more powerful and attractive solution is to use a dedicated device such as the I/O processor (IOP). The Am8089 IOP contains two independent channels with high speed DMA capability and its instruction set is extensively I/O-oriented. The bus organisation of the IOP is flexible so that data transaction is possible between 8-bit organised I/O devices and 16-bit wide memory or vice versa.

Fig. 6 illustrates such a system architecture for high-speed analogue I/O processing. Before being able to perform I/O tasks, the IOP must be initialised by the 8086/88 CPU which is configured in maximum mode. The system bus, is separated from the local bus to allow for parallel processing. The IOP's channel program may reside in the local memory space to further reduce system bus contention. The ADC may be initialised by the IOP. Data transaction from the ADC, DAC and local memory is then confined by the IOP to within the local bus. DMQ1 and DMQ2 are DMA request lines associated with the ADC and the DAC. DMA transfers may be terminated externally via EXT1 and EXT2.

Without a S/H device, the analogue input signal frequency range that can be effectively converted digitally by the Am6112 is very limited. This can be seen from the relationship, which defines the maximum slew rate of the sinusoidal input signal to the device:

$$\frac{dV}{dt} = 2\pi f V_{max}$$

$$f = \frac{dV}{dt}\left(\frac{1}{2\pi V_{max}}\right)$$

where dV is the maximum error allowed during the conversion, dt is the aperture time (t_A) of Am6112, and $2V_{max}$ is the maximum peak-to-peak amplitude of the sinusoidal input signal. The aperture time of the device is also its conversion time (t_C) which has a minimum value of $3\mu s$. Therefore, for $2V_{max} = 10V$, $dt = t_C = 3\mu s$, and $dV = 1/2LSB = 1.22mV$, $f = 12.94Hz$. By adding S/H device with an aperture time of 10ns., f becomes 3.88kHz.

Adding a S/H device decreases the throughput rate of A-to-D conversion process. This is because it takes time, defined as acquisition time (t_{ACQ}), for the S/H device to track the input signal to its specified accuracy.

Another factor that degrades throughput rate is the hold time, t_H. This is defined as the delay from the instant a hold signal is active to the time the analogue input signal settles to within a specified error limit. It is during this period that the analogue signal must be stable so that it can be converted into digital format to its desired accuracy. The typical figure for t_H is 100ns with 12-bits of accuracy.

The degradation of throughput rate due to t_{ACQ} can be minimised by putting the S/H device in sampling mode as soon as the ADC cycle is done. By the time another conversion cycle is to be started, the S/H device will have already tracked the analogue signal.

The design of high-resolution or low-level analogue/digital systems must include a thorough analysis of conditions that can degrade overall system accuracy. Consider the source impedance of the device driving the input of the ADC. The input impedance of the converter is modulated during the conversion cycle because currents of different values appear at the node where the ADC presents itself as a load to the S/H device. The output impedance of the S/H must be sufficiently low at all operating frequencies to avoid errors.

In addition, the S/H device output must settle to the required voltage level before a bit current comparison can be made. This means that the output transient response of the S/H must be able to cope with 1/2 clock period settling time requirement for each bit current comparison. For a 4MHz clock input to the converter this means a settling time to 12-bit accuracy within 125ns.

Another area that needs to be looked at carefully is the anti-aliasing filter requirement in the general case of a sampled data system where the input signal to be processed has several frequency components. An example of this is speech signal. The anti-alising filter rejects, as much as possible, any undesirable frequency components resulting from the sampling process. It also limits the noise components of the input signal. The low-pass (LP) filter that implements the anti-aliasing function should have maximally flat magnitude (MFM) characteristics. The well-known Butterworth filter characteristics are a special class of MFM filters exhibiting the least amount of ripple in the pass band which can introduce inaccuracies if the filter is not properly designed.

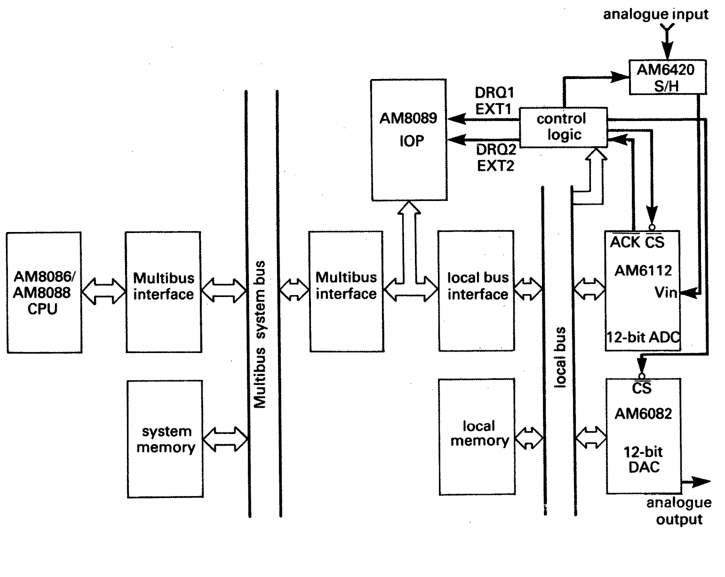
The sampling rate of a single channel sampled data system is determined by some of the dynamic characteristics of the analogue/digital components. A sampled data system may be classified as open loop, as in data collection, or closed loop, as applied in digital control systems. In general, the sampling frequency is determined as:

$$f_S = \left(\frac{1}{t_{ACQ} + t_H + t_C}\right) + \Delta t$$

where Δt is additional system delay. Δt can be zero in open loop systems and is variable in closed loop systems. Therefore, in a closed loop system, Δt is the determining factor that establishes the sampling rate requirements.

If the Am6112 is applied in a data collection system when $t_{ACQ} = 2\mu s$, $t_H = 100ns$, $t_C = 3.3\mu s$ (4MHz clock rate for the ADC), and $\Delta t = 500ns$, then $f_S = 169.5kHz$. Clearly, an input signal with highest frequency component of 4kHz can be sampled forty times easily. With the fast conversion time of the device, some of the S/H and anti-aliasing filter requirements can then be relaxed. This means a much more cost-effective design. □

Fig. 6: High-speed I/O processing.



Circle no. 402

Microprogrammable chips blend top performance with 32-bit structures

Broken down into 32-bit functional blocks instead of being sliced into multiple-bit sections, five VLSI bipolar chips match a supermini's speed.

Designers of systems and subsystems for high-speed computation, intelligent peripheral control, and array and digital signal processing typically need higher performance than standard microcomputer parts can deliver. The required precision, speed, and virtual memory support has to some degree been supplied by dedicated VLSI components that are customized for particular applications. Yet an overwhelming need still remains for a set of building blocks that can bring extremely high performance to a large assortment of applications.

A new approach extends the bit-slice concept to 32 bits and also satisfies system designs that require cycle times of less than 100 ns. With a family of five VLSI chips, designers of microprogrammed systems can count on cycle times of 70 to 80 ns, using merely a handful of components. The building blocks for 32-bit systems functionally partition the chips and separate the register file from the rest of the data path.

Paul Chu and **Bernard J. New**
Advanced Micro Devices Inc.

Paul Chu is now department manager of programmable processors in the product planning division of Advanced Micro Devices in Sunnyvale, Calif. He holds several patents for microprogrammable devices and has a BSEE and an MSEE from Stanford University.

As product planning manager for array processors at AMD, Bernard J. New is responsible for conceiving and defining arithmetic computing devices. The holder of a BSc (Hons) in electronic engineering from England's University of Birmingham, New has two patents on Am29500 products.

The following two articles first explore the key members of the Am29300 family and then focus on a floating-point processor, which is the first chip scheduled for sampling. Details are given on how to use the chip and other devices in the series to build a fast Fourier transform computer, as well as more general-purpose digital signal-processing circuits.

The Am29300 family addresses the problem of fault detection through an interlocking checking scheme—parity and master-slave. Byte parity is generated, stored, and then checked on all data-path elements as a means of detecting interconnection failures. Moreover, to verify certain functions, the master-slave operating mode permits two units to be connected in parallel, with one unit actually handling the computation and the other checking the result cycle by cycle.

Detecting a fault triggers an interrupt at the microinstruction level. Unlike previous redundant schemes, no specialized software is required. Furthermore, communication among the redundant functional units causes no system degradation.

The five chips form a strong foundation for any system designer's work. For instance, a 16-bit sequencer can handle interrupts and traps at the microinstruction level. There is also a combined ALU and shifter that internally supports variable byte and bit fields. Together with the ALU-shifter chip, a true dual-port register file, organized as 64 words by 18 bits, can build a basic system. The register file, designed for simultaneous read and write accesses, is separated from the data-path elements, thereby avoiding the problem of addressing an internal register file differently from external memory. The benefits of that separation are uniform register addressing and unlimited depth expansion.

Two accelerator chips—a floating-point processor and a parallel multiplier—can be added to the basic system to raise the number of functions and cut processing time. The 32-by-32-bit parallel multiplier can, on successive cycles, expand to 64 by 64 or 128 by 128 bits, without help from external logic. For its part, the math chip can tackle single-cycle addition, multiplication, subtraction, and conversions—all in single-precision IEEE or DEC formats.

Because of functional partitioning, a three-bus flow-through architecture was chosen as the data path. For maximum bus accessibility, all data-path elements—the integer processor and the parallel multiplier, for example—share two operand and one result bus. The flow-through architecture not only transfers data extremely quickly but also avoids the complex timing control needed to turn around bidirectional buses. Above all, the simplicity of the three-bus architecture allows these components to be configured in a variety of ways to optimize micro-architectures for different jobs.

Bipolar building blocks deliver supermini speed to microcoded systems

As CMOS processes start to encroach on the performance of bipolar circuits, bipolar technology is taking the next step to keep itself in the lead for the highest speed systems. A family of five bipolar VLSI computational circuits—fabricated with a scaled, ion-implanted, oxide-isolated process and three levels of metal interconnections for high density—provides a set of functionally partitioned microprogrammable VLSI building blocks for systems such as superminicomputers, digital signal processors, high-speed controllers, and many others. The modularity of the system functions ensures that the chips can meet the performance requirements of a general-purpose superminicomputer, as well as those of an image processor, which are radically different from each other.

Included in the family are three parts that form the core of a general-purpose microprogrammed system: a 32-bit arithmetic and logic unit (ALU), a 16-bit microprogram sequencer, and a 64-by-18 four-port, dual-access RAM. And, for systems that do a large number of multiplications or floating-point

Dhaval Ajmera, Øle Moller, and **David Sorensen**
Advanced Micro Devices Inc.

Since the beginning of last year, Dhaval Ajmera has been a design engineer in product planning at Advanced Micro Devices in Sunnyvale, Calif. He holds an MSEE from the University of Florida.

Øle Moller is also a design engineer in AMD's product planning operation. He holds an MSEE from the Technical University of Denmark.

Another engineer in product planning, David Sorensen specializes in programmable processors. He holds a BSEE from Arizona State University.

operations, two performance accelerators—a 32-by-32-bit multiplier and a 32-bit floating-point processor will be available to tie onto the buses (see Design Entry, p. 246).

The chips offer high performance, a flexible architecture, and microprogrammability, and even address the problem of fault detection for data integrity. These circuits can thus support an extremely fast microcycle—about 80 ns (projected). That high speed is the result of several design considerations: Each part is designed internally with emitter-coupled logic but has TTL-compatible inputs and outputs. Second, more power was allocated to the logic circuits used in the critical paths than for logic in the noncritical paths on each chip, to maximize the speed. Third, by integrating highly specialized logic on chip it is possible to execute very complex operations in a single cycle.

The microprogrammability of this chip set offers several benefits to the system designer. It provides a structured and systematic approach for implementing the control mechanism of the system, and like the bit slices, it allows the instruction set to be customized to suit the designer's application (see "Architectural Limitations of Bit Slices," opposite). And several versions of the initial design can be tested, or current designs can be enhanced simply by changing the microcode.

Thus, the functionally partitioned Am29300 family overcomes all of the performance penalties of bit-slice structures, while maintaining its ability to form a wide variety of architectures. Even though the chips are designed to work together as a family, each can also be used independently in an application that requires its unique capabilities.

Pipelines are out

The flexibility of the Am29300 family is largely due to a decision not to place pipeline stages within the functional blocks. Not including the pipeline registers inside incurs some off-chip delays. This is a small price to pay to allow system designers to optimize the pipeline structure for their individual needs. Moving the register file out of the functional block for the ALU also slows things down. At the same time it does not force a fixed register size on the user, enabling systems to be created with dedicated registers, register windows, or register banks—all with neither fixed depth nor width.

Additionally, the high level of integration helps eliminate the propagation delays often encountered when signals must go from chip to chip. The use of VLSI also results in fewer parts at the system level, which, in turn, conserves power (usually many watts in the case of bipolar systems) and board space. Lastly, a complete 32-bit solution is provided for applications that require increased precision for arithmetic operations, high memory bandwidth, and a

Architectural limitations of bit slices

The limited performance of bit-slice circuits can be improved by increasing the width of the slices. That higher level of integration results in higher performance by reducing the number of off-chip delays while preserving the flexibility that has made bit-slice systems so attractive. However, as higher levels of integration become possible, two inherent problems with bit-slice architectures will limit their ultimate speed. The first involves the off-chip delays inherent in cascading. For example, the carry chain is usually the slowest path of an ALU. Breaking this chain between slices introduces off-chip delays into the critical path.

The second problem is that the functional needs of many systems do not slice well. Barrel shifters and prioritizers are especially difficult to cascade. Unfortunately, the ability to perform N-bit shifts and locate the position of leading 1s are of greatest importance in applications that require heavy number crunching and manipulation of data fields, such as image processing, graphics, database management, and controllers. These are precisely the applications whose need for speed forces the use of bit-slice devices. The system performance is compromised not only because these operations must be done bit by bit, but also because many high speed algorithms cannot be efficiently implemented.

Microprogrammable 32-bit chips

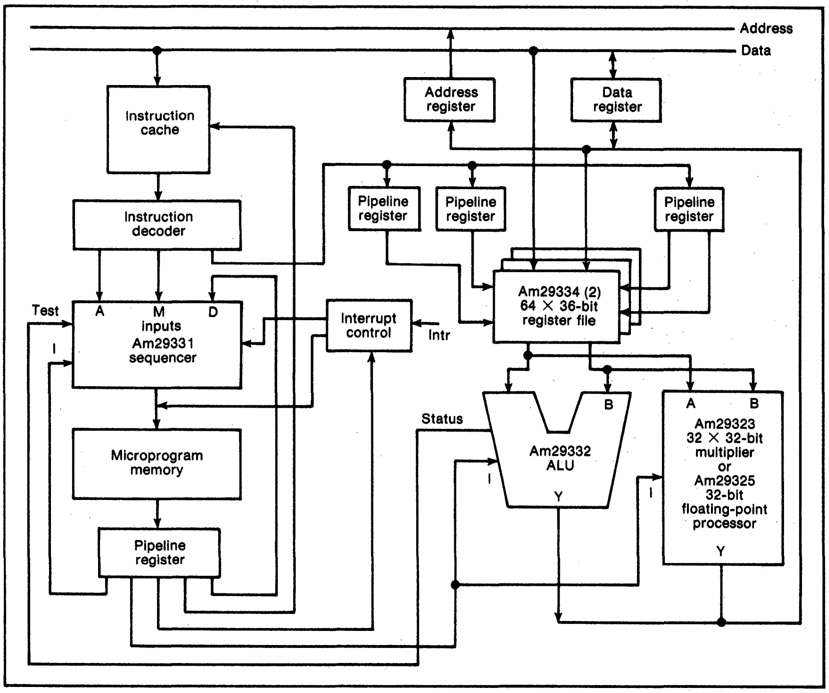
large addressing capability (4 billion bytes) to support virtual memory systems (Fig. 1).

The performance of a system depends, not just on its raw computing speed, but on its ability to respond to events such as interrupts and traps. For example, the Am29331 sequencer responds to both interrupts and traps at the microprogram level very quickly, and its response is completely transparent to the interrupted microroutine. Also, the Am29332 ALU indirectly supports the handling of these events by allowing its internal state to be saved or restored.

The Am29332, a noncascadable 32-bit-wide, ALU, provides fast number crunching, high data transfer rates, and powerful bit-manipulation capabilities. Intended to be used with the Am29334 dual-ported RAM, which serves as an external register file, the ALU has two 32-bit input buses (DA and DB) and one 32-bit output bus (Y).

Internally, the device has a 32-bit data path that interconnects its various functional blocks. These blocks include various shifters and multiplexers, a mask generator, a funnel shifter, the ALU proper, a priority encoder, a parity generator and checker, a master-slave comparator, and the status and Q registers (Fig. 2). The ALU proper has three 32-bit inputs: R, S and M. The R input comes from the funnel shifter, the M input from the mask generator, and the S input from a variety of sources —the DA or DB buses, status register, or the Q register.

The power and flexibility of the Am29332 comes partly from its ability to perform operations on various data types. It can operate on



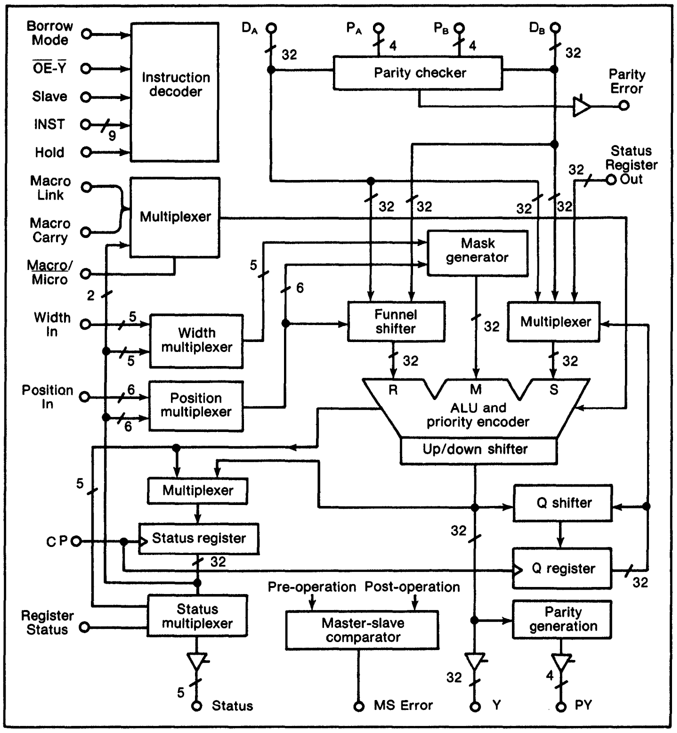
1. A conventional CPU, built with Am29300 building blocks, forms the focal point of an extremely compact system that cycles as fast as 80 ns.

variable bytes, variable-length bit fields, or single bits. This is made possible by the internal mask generator, which creates a 32-bit mask for each instruction (with no time overhead). The mask is used as an additional operand in each instruction to allow the operation on only selected data widths.

The type of mask generated depends on the type of instruction. For instructions that operate on variable bytes (1, 2, 3 or 4 bytes) the mask is a fence of 1s (bit 0 aligned) for all low-order selected bytes with a fence of 0s for all high-order unselected bytes. Instructions that operate on variable-length bit fields require a mask that is a string of contiguous 1s for all selected bit positions and 0s for all unselected bit positions. In cases where the field exceeds the 32-bit boundary, the mask does not wrap around, thus allowing operation on a contiguous field across a word boundary. For instructions that operate on a single bit, the mask is a 1 for the selected bit position and 0s for the other unselected bits.

For most single-operand instructions, the unselected bit positions pass the corresponding bits of the operand unmodified. For most two-operand instructions, the unselected bit positions pass the corresponding bits of the operand unmodified on the DB input. Thus, for two-operand instructions the mask allows the merging of two operands in a single cycle. In addition to being used internally, the mask can be sent out over the Y bus, permitting the generator to be used as a pattern generator for testing purposes.

To speed various mathematical and logical operations, many circuits have started to in-



2. To connect its various internal functional blocks, the Am29332 ALU employs a 32-bit bus. Among the chip's major features are a 64-bit funnel shifter, parity checking and generation, and a basic 32-bit ALU that has three input ports. The processor also has three 32-bit ports through which it transfers data into and out of the chip.

clude a barrel shifter, which has an N-bit input and an N-bit output. The barrel shifter would be used to shift or rotate the operand either up or down from 0 to N bits in a single cycle. Such high-speed shifting is very useful in operations such as the normalization of a mantissa for floating-point arithmetic or in applications in which the packing and unpacking of data are frequent operations.

However, a more useful circuit is a funnel shifter, which can be thought of as having two N-bit inputs and one N-bit output. Just such a circuit (with 32-bit-wide ports) was included on the 29332. The circuit can perform all the operations of a barrel shifter with capabilities extended to two operands instead of one. In addition, it can extract a 32-bit contiguous field across its two operands, a function very useful in several graphics applications. And any of its operations can be followed by a logical operation, with both completed in a single cycle.

Setting the priorities

Prioritization, useful to control N-way branches, perform normalizations, and in graphic operations such as polygon fills, can readily be handled by the ALU chip. The built-in priority encoder sends out a 5-bit binary weighted code that signifies the relative position of the most-significant 1 from the most-significant bit position of the byte width selected. That allows prioritization on either 8-, 16-, 24-, or 32-bit operands. The priority encoder output can be passed on to the Y bus or stored in the status register.

If, for example, prioritization is used to normalize a mantissa during a floating-point arithmetic operation, it requires two cycles. In the first, the mantissa is prioritized to determine the number of leading 0s that need to be stripped off. In the next cycle, the mantissa is shifted up by the amount specified by the priority encoder output.

Relevant information for each operation performed by the chip is stored in the 32-bit status register after each microcycle. Each byte of the status word holds different information. The least-significant byte holds the position specifier. The next most-significant byte holds the width specifier and three other bits that are used to test the comparison of unsigned and signed operands. The next byte contains the Carry, Negative, Overflow, Link, Zero, M and S flags. The M flag stores the multiplier bit for multiply or the sign compare bit for signed division, and the S flag stores the sign of the partial remainder for unsigned division. The most significant byte stores the nibble carries for BCD operations.

The states of the Carry, Negative, Overflow, Link and Zero flags are available on the status pins, and the status multiplexer allows the user to select either the status of the previous instruction (register status) or the status of the current instruction (raw status) to appear on the status pins. The raw status could be used to update an external macro status register. This also allows branching at either the micro- or macro-level.

The Q shifter and Q register are primarily used to assemble the partial product or partial quotient in multiplication and division operations. Variable bytes of the status and Q register can either be loaded via the DA and DB inputs or can be read over the Y bus. Thus saving and restoring of the registers allows efficient interrupt handling after any microcycle. It is also possible to inhibit the update of both these registers by asserting the Hold pin.

Powerful and orthogonal instructions

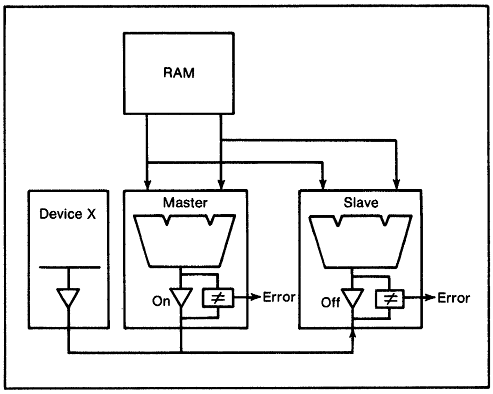
The power of the ALU chip's instruction set comes directly from the integration of several functional blocks mentioned earlier. The commands are symmetrical as well as orthogonal, to make it easier for a compiler to generate efficient code. Thus, any operation on the DA input is also possible on the DB input, and each instruction is completely independent of its data type.

Three-fourths of the instruction set consists of variable byte-width (one, two, three or four) operand instructions. The byte-width is selected by two bits in the instruction. For these operands, the instruction set supports all conventional arithmetic, logical and shift operations. Arithmetic operations can be performed on both signed and unsigned binary integers.

Additionally, the instruction set supports multiprecision arithmetic such as addition with carrying and subtraction with carrying or borrowing. For all subtract operations it provides the convenience of using borrowing instead of carrying by asserting the borrow pin. In this mode the carry flag is updated with the true Borrow. To allow efficient execution of macroinstructions the chip contains a Macro mode pin. When the chip asserts this pin, it allows the external Macro-Carry and Macro-Link bits instead of their microcounterparts to participate in the operation.

Instructions that execute algorithms for the multiplication and division of signed and unsigned integers are multiple cycles are also provided. For multiplication, the circuit supports the modified Booth algorithm, yielding two product bits in one cycle. Both single-precision and multiprecision division of signed and unsigned integers are supported at the rate of one quotient bit in every cycle.

Besides binary integers the instruction set provides basic arithmetic operations for binary-coded decimal (BCD) numbers. By operating directly on the decimal numbers created in most business applications, significant processing time is saved by eliminating the need to convert from binary to BCD and vice versa. Also, the round-off errors involved in converting from one base to the other are eliminated.



3. To help ensure system integrity, two Am29332 processors can be set for master and slave operation. Both chips perform the same operation in parallel, and any difference in their results is flagged as an error. The master also checks its internal result against the data on the output bus to make sure that no other device (such as device X) is turned on at the same time.

The last group of instructions was created to support variable-length bit fields (1 to 32) and single-bit operands. The position and width of the field can be specified by either the position and width inputs or by fields in the status register, thereby saving bits in the microcode. Most of the time, the position and width are determined dynamically. It is therefore difficult to supply them via the microinstructions. For single bit operations only the position specifier is needed.

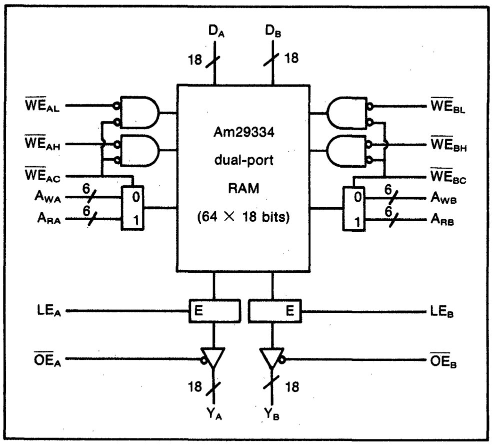
Bit-manipulation instructions include setting, resetting, or extracting a single bit of the operand or the status register. Logical operations on either aligned or nonaligned fields in the two operands include OR, AND, NOT and XOR. In the case of nonaligned fields it is assumed that at least one of the fields is aligned to bit position 0. It is also possible to extract a field from one operand and insert it into another operand or extract a field across two operands.

Enhancing system integrity

The growing need for data integrity has been addressed at both the system and the chip level by including hardware for fault detection. During calculations, byte-wide even parity is generated for the data result by the ALU and stored with the data in the external RAM. Byte-wide even parity is also checked at the ALU inputs and any error is flagged.

Even parity is specifically used to check for a floating TTL bus. Thus, all interchip connections are checked out. In addition, hardware for functional verification is also provided on the sequencer and the ALU functional verification can be implemented by using two similar devices in the master and slave mode (Fig. 3). In that setup, both chips perform the same operation, with any difference in their outputs being flagged as an error. The slave-mode chip's bidirectional buses operate in their input mode, allowing the master to compare its own internal result with that of the slave on every cycle. Additionally, the master checks the output bus to make sure that no other device is turned on at the same time.

As mentioned earlier, the ALU architecture was designed to use an external register file. Keeping the file external to the chip permits the user to expand it to meet any system need. The Am29334, a high-speed 64-word-by-18-bit dual-access RAM, provides two independent data input ports and two independent data output ports (Fig. 4). Each port can be read from or written to using the separate inputs and outputs. The two accesses are independent except for the case when simultaneous write operations are done to the same word—in which case the result is undefined. The read address inputs and the write address inputs of each side are separate in order to save the cost and time delay of external multiplexing between a read address and a write address.



4. The dual-access RAM serves as an external register file for the arithmetic processor chip. The Am29334 holds 64 words, each 18 bits long. Two chips are often connected to build a RAM block with four data outputs, two data inputs, and six address lines. Each port of the RAM can be independently accessed to read or write.

The word width of 18 bits allows the RAM to store two bytes plus a parity bit for each. Each side has separate write enable for the lower and upper nine-bit bytes and a common write enable that also switches the address multiplexer. The actual write is delayed internally to allow the write address to set up internally before writing starts.

It is possible to build a RAM with four data outputs, two data inputs and six addresses by using two dual-access RAMs and on each side connecting the data input, write address and write enables of one RAM in parallel with the corresponding inputs of the other RAM. This expanded RAM may be used in concurrent processing applications in which an ALU and an adder (which generates the address) do their computations—this yields a result and an address in parallel. The two values can then be fed simultaneously to the multiport memory.

The sequencer controls the show

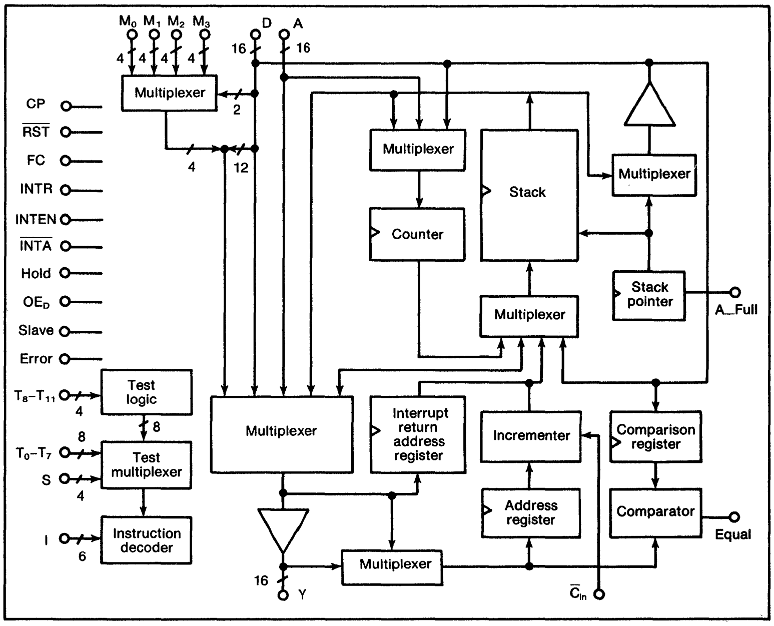
The cycle time of the microprogrammed system is dependent on both the control path (i.e., sequencer and microprogram memory) and the data path (i.e., register file and ALU). Traditionally, the system bottleneck has been the control path, especially the ciritical paths associated with conditional branching. Special care has been taken in the design of the Am29300 family to balance control and data-path timing.

A key device contributing to the improved control-path timing is the Am29331 16-bit microprogram sequencer. It is designed for high speed, and that speed has been attained by the elimination of functions that would slow down the microaddress selection and by including the test logic and the test multiplexer in the sequencer (Fig. 5). As in most previous generation sequencers, the address register, the incrementer, the address multiplexer, the stack, and the counter are standard functions. The sequencer has multiway branch instructions that allow 1 of 16 consecutive addresses to be selected as the branch target in a single cycle.

The address register in most other sequencers is called a program counter, but this name is not correct if a strict definition is applied. In the Am29331, the incrementing counter is placed after the address register, which thus allows for the handling of traps. The stack stores return addresses, loop addresses and loop counts. It has 33 levels to permit the deep nesting of subroutines, loops and interrupts. An output, Almost Full (A-Full), indicates when 28 or more of the levels are in use.

Available for use in iterative loops, the counter can be loaded with an iteration count at the beginning of a loop, and the count is tested and then decremented at the end of the loop. The loop is terminated if the count is equal to one; otherwise a jump to the beginning of the loop is executed.

There are three buses that carry microaddresses. The bidirectional D bus can be connected to the pipeline register, providing branch addresses or loop counts, or used for two-way communication with the data processing part of the system. The A bus, called an alternate bus, can be connected to a mapping PROM to provide starting microaddresses for instructions in a computer. The Y bus sends out



5. To aid in handling trap operations, the incrementer is placed after the address register in the Am29331 microsequencer. Additionally, the chip has a 16-bit address bus, which enables it to access up to 64 kwords of control memory and handle interrupts and multiple-path branches.

selected microaddresses to the microprogram memory and accepts interrupt or trap addresses if interrupt or trap is employed.

Four sets of 4-bit multiway inputs provide a simultaneous test capability of up to 4 bits. And, one way to use those inputs would be to decode mode bits in changing positions in macroinstructions. The four select lines select 1 of 16 tests to be used in conditional instructions. There are twelve test inputs. Four of these may be used for C (Carry), N (Negative), V (Overflow) and Z (Zero), generating internally the tests C+Z, $\overline{C}$ + Z, N XOR V, and N XOR V+Z, which are used for comparison of signed and unsigned numbers.

Relative addressing was the only somewhat useful function that was removed in order to maximize speed. The sequencer supports interrupts and traps with single-level pipelining, but may also be used with two levels of pipelining in the control path. It has a 16-bit-wide address path and cannot be cascaded, which thus limits the addressable memory depth to 64 kwords of microcode. That, however, is sufficient for the vast majority of applications—a typical computer, for instance, that has a microprogrammed instruction set, might use only about 1 to 2 kwords. However, for systems in which the microprogram is the sole program level, its size is generally larger.

Microprogram interrupts supported

The Am29331 sequencer supports interrupts at the microprogram level. Like polling, interrupts handle asynchronous events. However, polling requires explicit tests in the microprogram for events, thus leading to long response times, lower throughput, and larger microprograms. Interrupts, on the other hand, have a response time equal to the cycle time of the system (approximately 80 ns), measured from the Interrupt Request input (INTR). The sequencer accepts interrupts at every microinstruction boundary when the Interrupt Enable input (INTEN) is asserted.

An actual interrupt turns off the Y bus driver and asserts the Interrupt Acknowledge output (INTA), which should be used to enable an external interrupt address onto the Y bus, thus driving the microprogram memory. The interrupt also causes the interrupt return address to be saved on the stack; this permits nested interrupts to be handled (Fig. 6).

The Am29331 is also the first sequencer that can handle traps. A trap is an unexpected situation caused by the current microinstruction, which must be handled before the microinstruction completes and changes the state of the system. An attempt to read a word from memory across a word boundary in a single cycle is an example of such a situation. When a trap occurs, the current microinstruction must be aborted and re-executed after the execution of a trap routine, which will take corrective measures.

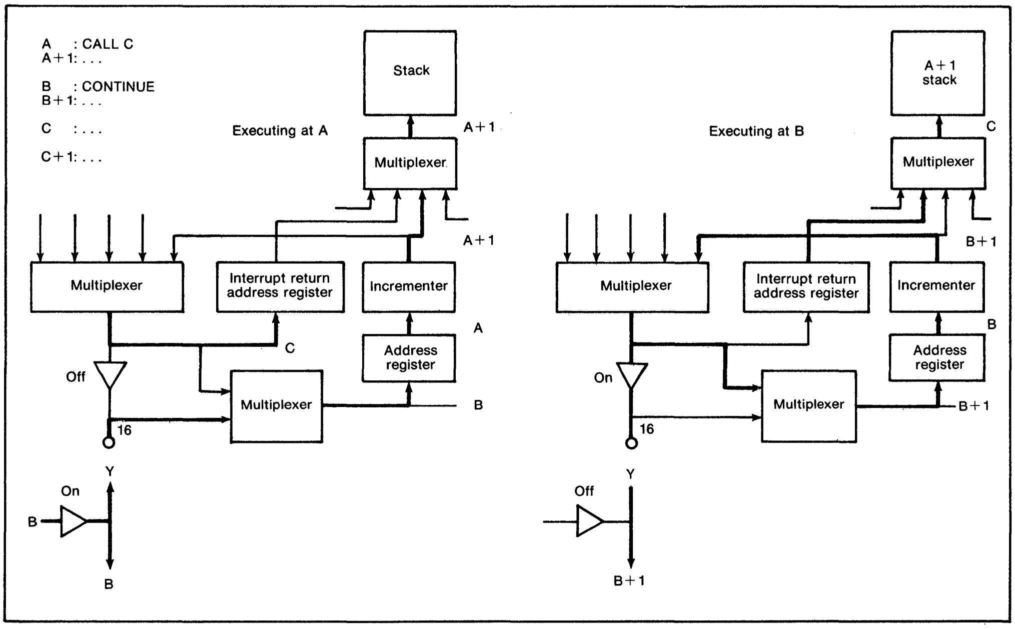
Execution of a trap requires that the sequencer ignore the current microinstruction and push the trap return address—the address of the ignored microinstruction—on the stack. The trap address must be transferred onto the Y bus at the same time. All this can be accomplished by disabling the carry-in to the incrementer ($\overline{C}_{in}$) and asserting the Force Continue input (FC) and the Interrupt Request input (INTR).

Also built into the sequencer is an address comparator, which allows detection of breakpoint in the microprogram. An output signal from the comparator indicates when the content of the comparator register is equal to the address on the Y bus. There is an instruction that loads the comparator register from the D bus and enables the comparator, which may later be disabled by another instruction.

Parallel microprocesses are useful when the system must deal with peripheral devices that are controlled at the microcode level. Normally only one processor is present and it must be time multiplexed between the concurrent operations that must be performed. When a process is suspended its private state must be saved, so that it can be restored when the process resumes execution. That, in turn, requires that the state of the sequencer be saved and restored, or each process must have its own sequencer that is active when the associated process is active. The first approach is the least expensive, but the second offers the advantage of shorter response time, because no time is spent on saving and restoring the state.

The Am29331 supports the first approach with its bidirectional D bus, through which the entire state, with the exception of the comparator register, can be saved and restored. The sequencer also supports the multiple sequencer arrangement, in which the three-state Y buses from the sequencers are tied together driving a single microprogram memory. One of the sequencers is active, while the remaining sequencers are put on hold by asserting their Hold inputs. The Hold input disables most outputs (the D bus synchronously), disables the incrementer, and enables an internal Force Continue. This effectively detaches the sequencer from the system and preserves its state.

The sequencer has a 6-bit instruction input that is internally decoded to yield a set of 64 instructions. There are 16 basic branch instructions, each in an unconditional version, a conditional version, and a conditional version with complemented test. In addition there are 16 special instructions like Continue and Push C (push counter on stack). The branching instructions handle jumps, subroutines, various kinds of loops and exits out of loops, and FC actually overrides the instruction inputs with a continue



6. Because it can accept interrupts at any microinstruction boundary, the sequencer responds faster than most other microprogrammed systems. For example, while the instruction at point A in memory is being executed, the sequencer is directed to point B. The only restriction on the programmer is that the first instruction of the interrupt routine cannot use the stack, since the interrupt return address is pushed onto it at the start of the procedure.

instruction. FC is useful in field sharing and support for writable microprogram memory.

The Am29331 is one of the few sequencers where the stack is accessible from outside through the bidirectional D bus. This indirectly allows access to the whole state of the sequencer except the comparator register. This is useful when testing the device, and during system debugging, in which, for example, the contents of the counter and the stack may be examined and altered. By including the troubleshooting instructions in the microcode, the sequencer may aid in debugging itself and the rest of the system. The access to the state is also useful for changing context or extending the stack outside.□

Single-chip accelerators speed floating-point and binary computations

Complex multiplication or floating-point mathematical operations are frequently needed in most computer systems, but in many cases, not often enough to warrant the added cost of dedicating CPU hardware to the computational job. To speed up the calculations, many systems, though, allow for accelerator boards or boxes that can perform such operations at several megahertz speeds or more.

Already, many silicon designers have developed chips to simplify the design of such subsystems—16-bit parallel multipliers fabricated in bipolar, CMOS or NMOS processes, and single-chip or multichip floating-point processors made with CMOS or NMOS have been available for some time. However, they are low-performance solutions to the problem, or in some cases, have limited application since they are intended for highly pipelined systems.

Now, the ability to handle 32-bit binary multiplication or 32-bit floating-point multiplication, addition or subtraction can be added to a system with just a single chip. The Am29323 is a 32-bit parallel multiplier that accepts two 32-bit inputs and can deliver a 64-bit product in a single clock cycle of 80 ns. Alternatively, performing floating-point operations, the Am29325 accepts two 32-bit inputs and delivers a 32-bit result in less than 125 ns. It can operate with numbers represented in either the IEEE (P754) or Digital Equipment Corp. floating-point formats and can convert numbers from one format into the other.

Both chips are part of the just unveiled Am29300 series of 32-bit computational elements (Design Entry, p. 230). The multiplier is ideal for computer systems that do floating-point operations only infrequently but must often perform high-speed integer calculations such as those required in image manipulation. The floating-point processor enhances systems used for fast Fourier transform and scientific calculations. Systems could even contain both accelerators if a high-performance, general-

David Quong and **Robert Perlman**
Advanced Micro Devices Inc.

David Quong is a product planning engineer with the digital signal processing and array processing group at Advanced Micro Devices in Sunnyvale, Calif. He received a BSEE from California State University in Sacramento.

Robert Perlman is a senior product planning engineer with the digital signal processing and array processing group. He obtained a BSEE from the Rensselaer Polytechnic Institute and an MSEE from the Johns Hopkins University, and has previously done design work in airborne digital signal processing at Westinghouse.

purpose system were built (Fig. 1).

To speed the flow of data into and out of the chips, both circuits were designed with two 32-bit-wide input ports and one 32-bit output port. But the similarities end there, since the chips perform vastly different operations on the data. A fairly straightforward design, the multiplier uses a full Booth-encoded array to deliver a 64-bit product to the output register (Fig. 2). The output register feeds a multiplexer that sends the result, 32 bits at a time, to the output port.

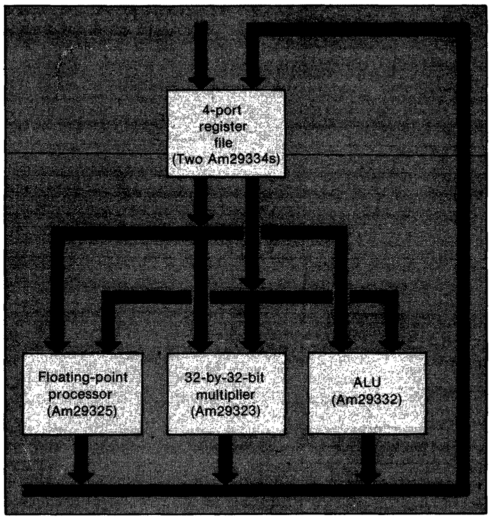
Double-precision operations can be done thanks to dual 32-bit input registers that are multiplexed into the multiplier array. A 67-bit partial-product adder allows new products to be summed with the contents of the output register. During this operation, the contents of the output register may be scaled by 32 bits, if necessary. Four partial products are formed and summed, and a temporary register assists in the scheduling of output transfers. The effective pipelining throughput in the double-precision mode is one 64-bit multiplication every four cycles. The accumulator can also support 96- and 128-bit multiplications. However, for such operations, input data must be repeatedly applied.

The input and output registers of the multiplier have independent control signals so that they can be optimally timed in pipelined systems. However, in unpipelined systems, the registers can independently be made "transparent" so that data encounters no delays when entering or leaving the chip. Like the other chips in the Am29300 family, the multiplier has parity checking and generating circuits to ensure system data integrity. And, the circuit offers a slave mode in addition to its normal mode—if two chips are tied together to operate in parallel with one set to operate in the slave mode, the circuits will generate an error flag if unequal results are obtained.

In the world of floating-point computations, several single-chip units, designed to be general-purpose math coprocessors for microprocessor systems have achieved close to microsecond operating speeds. However, to achieve higher throughput rates, several recently announced two-chip sets have cut that speed by a factor of 10, achieving data throughput rates of 10 MHz for pipelined operations. But, if operated in nonpipelined systems, these chips lose considerable speed—often by a factor or two or three—since data must ripple through the stages of pipeline registers.

To cut the data delays, the Am29325 took a direct approach and eliminated all the pipelining. It is the first floating-point processor to contain a 32-bit floating-point adder/subtractor, multiplier, and flexible 32-bit wide data path on a single chip (Fig. 3). Additionally, support for division operations is included on the chip as well as a status flag generator.

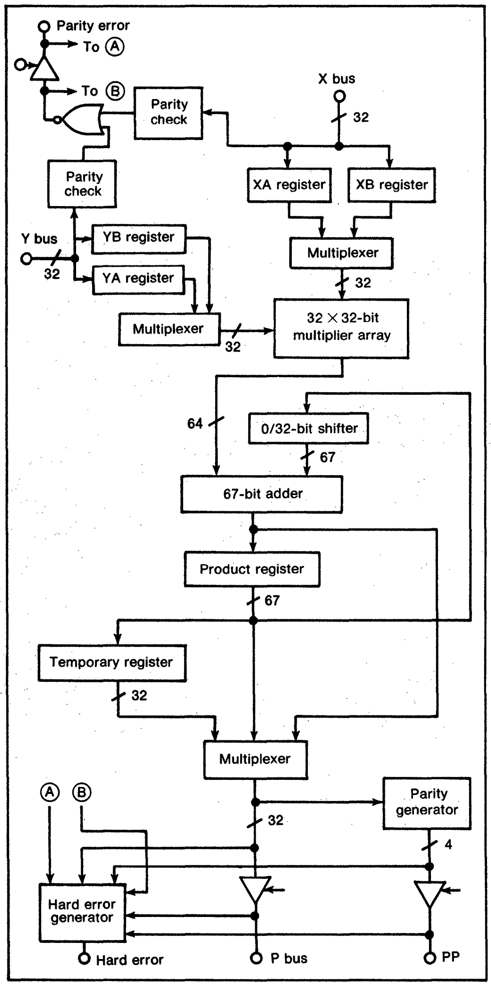
Fabricated with the IMOX-S bipolar process and three levels of metal interconnections and


4-port register file (Two Am29334s)
Floating-point processor (Am29325)
32-by-32-bit multiplier (Am29323)
ALU (Am29332)

1. The 32-bit multiplier and the 32-bit floating-point processor can be used together in a system. Either chip also functions without the other if just one of the capabilities is needed.

32-bit math accelerators

housed in a 144-lead pin-grid-array package, the Am29325 can replace one to two boards of SSI and MSI logic typically used in general-purpose computers, array processors and graphics engines, to provide high-speed floating-point math capability. When used in concert, the on-chip functions will meet the computational and data-routing needs of these and many other applications.



2. Surrounding the 32-by-32-bit multiplier array on the Am29323 are multipliers for the two 32-bit input buses, which permit 64-bit multiplications to be done in just four cycles. The multiplier checks parity on the input data and generates parity bits for the output result.

Integrating these functions into a single device greatly reduces data routing problems and minimizes processing overhead that would otherwise be incurred when shuffling data on and off the chip. The internal data path is ideally suited for multiplication and accumulation, Newton-Raphson division, polynomial evaluation, and other often-used arithmetic sequences. Placing the data path on chip also dramatically reduces the number of ICs needed to interface the device to the rest of the system.

The three-port floating-point arithmetic unit at the chip's core can perform any of eight instructions in a single clock cycle. The absence of pipeline delay in the arithmetic unit means that the result of an operation is available for use as an input operand in the very next operation, a crucial feature when performing algorithms with tight feedback loops. Instructions and other operating modes are selected with dedicated input signals, an approach ideally suited to microprogrammed environments. The device easily interfaces with a variety of 16- and 32-bit systems using one of three programmable bus modes.

Delving into the operation

At the heart of the arithmetic unit are a high-speed adder-subtracter, a 24-by-24-bit multiplier, an exponent processor, and other logic needed to implement the floating-point operations. Two input ports, R and S, provide operands for the instruction to be performed; the result appears on port F. One of eight instructions is selected by placing a 3-bit code on lines I_0, I_1, and I_2. The first three instructions—$R + S$, $R - S$, and $R \times S$—operate on both input operands; the remaining instructions need only one input operand.

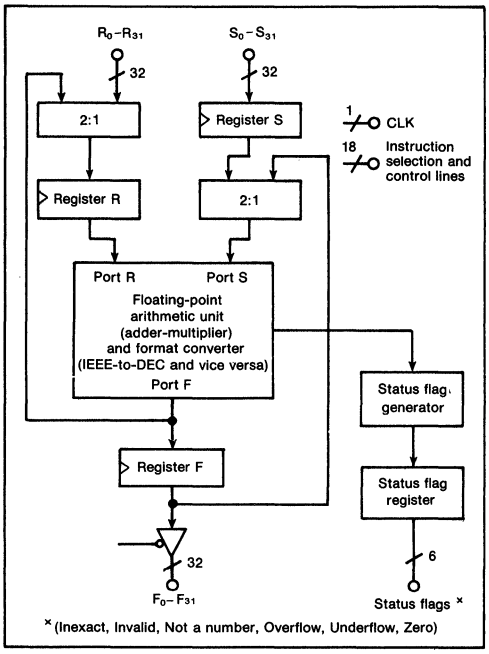
The fourth instruction, $2 - S$, forms the core of the Newton-Raphson division algorithm, in which the quotient A/B is calculated by first evaluating 1/B, then postmultiplying by A. The reciprocal value 1/B is derived by using an external lookup table to provide an approximation of 1/B; this approximation is refined using the iterative equation:

$$x_n = x_{n-1}(2 - Bx_{n-1}),$$

32-bit math accelerators

where x_n is the nth approximation of 1/B.

Once B and the approximation of 1/B are loaded into the Am29325, the approximation is refined using a sequence of R × S and 2 − S instructions; no additional I/O operations are needed for reciprocal refinement. The remaining four instructions perform data format conversions. Instruction INT to FP converts a 32-bit, two's complement integer to floating-point form, useful when processing data initially generated in fixed-point format; conversion from floating point to integer format is handled by instruction FP to INT. Two other instructions convert between IEEE and DEC floating-point formats.



R_0-R_{31}
32
S_0-S_{31}
32
2:1
Register S
1
CLK
18
Instruction selection and control lines
Register R
2:1
Port R
Port S
Floating-point arithmetic unit (adder-multiplier) and format converter (IEEE-to-DEC and vice versa)
Port F
Status flag generator
Register F
Status flag register
32
6
F_0-F_{31}
Status flags ×
× (Inexact, Invalid, Not a number, Overflow, Underflow, Zero)

3. Also using separate 32-bit buses for the inputs and output, the AM29325 floating-point processor handles either IEEE or DEC formatted data and can translate between formats, if necessary.

The arithmetic unit recognizes two single-precision floating-point formats—the IEEE format as specified in proposed standard P754, draft 10.0, or the DEC format used in VAX minicomputers. The eight instructions can be performed using either format; the desired format is selected with the IEEE/DEC pin on the processor chip. The formats are broadly similar—each has an 8-bit biased exponent, a 24-bit significand comprising a 23-bit mantissa appended to an implied or "hidden" most-significant bit (MSB), and a sign bit.

There are, however, a number of subtle differences. The IEEE format has an exponent bias of 127 and a binary point placed to the right of the hidden bit, while the DEC format has an exponent bias of 128 and a binary point placed to the left of the hidden bit—these variances result in a slightly different range of representable values. Each format has its own set of operands reserved for special uses. The IEEE format reserves operands to represent non-numerical values (referred to as Not a Number, or NaN), $+\infty$, $-\infty$, and plus and minus 0; the DEC format reserves only two types of operands to represent non-numerical values and 0. In addition to format differences, there are a number of minor differences in the manner in which operands are handled during the course of a calculation. These differences are automatically accounted for when the desired format is selected.

The need for rounding

When performing a floating-point operation, it is sometimes possible to generate a result whose value cannot be precisely expressed as a floating-point number. If, for example, the single-precision floating-point values 2^{23} and 2^{-1} are added, the infinitely precise result, $2^{23} + 2^{-1}$, cannot be represented exactly in the single-precision floating-point format. Some means, then, must be provided for mapping the infinitely precise result of a calculation to a representable floating point value. The arithmetic unit implements four IEEE-mandated rounding modes to afford the user some flexibility when performing this mapping; the desired rounding mode is selected with signals RND_0-RND_1.

Of the four modes, the round-to-even mode is most often used; it maps the infinitely precise result of an operation to the closest representable floating-point value. The round-toward $-\infty$ mode maps to the nearest representable value less than or equal to the infinitely precise result; similarly, the round to $+\infty$ mode maps to the nearest value greater than or equal to the infinitely precise result. A fourth mode, Round toward zero, maps to the closest representation whose magnitude is less than or equal to that of the infinitely precise result. As one would expect, if the infinitely precise result of an operation is representable in the floating-point format, it passes through the rounding operating unchanged, regardless of rounding mode.

As the result of an operation, various status flags are set or reset by the status flag generator. Six flags are used to note the occurrence of overflow, underflow, zero, not-a-number, invalid, or inexact conditions. Because the flags are generated as the operation is performed, the user can greatly reduce processing overhead that would otherwise be needed to test the results of operations. The flags are fully decoded, minimizing the amount of hardware needed to interpret them.

Flagging the status

Four of the status flags report exception conditions stipulated in IEEE standard P754. The Invalid flag indicates that an input operand or operands are invalid for the operation to be performed. The Underflow and Overflow flags are active when a result is too small or too large for the operation's destination format. The fourth exception flag, Inexact, tells the user that the result of an operation is not infinitely precise. Although these flags are primarily an adjunct to operation in the IEEE format, they also produce valid results when the DEC format is selected. The Am29325 generates two additional flags not provided for in the IEEE standard. Flags Zero and NaN identify zero-valued or nonnumerical results for both IEEE and DEC formats.

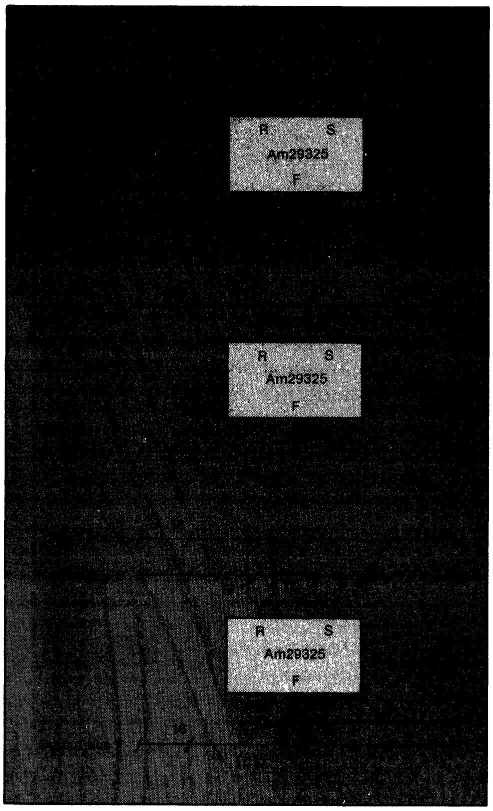
A floating-point processor whose arithmetic unit performs millions of operations per second can maintain that operating speed only if the correct operands can be routed to the arithmetic unit at that rate; if not, the specification is meaningless. To meet this crucial requirement, the core of the Am29325 is supported by a 32-bit data path comprising two input buses, a three-state output bus, and two data feedback paths. These data paths give the user the means to get the operands to where they are needed without devouring extra clock cycles.

Data enters through input buses R_0-R_{31} and S_0-S_{31}; results exit through three-state output bus F_0-F_{31}. Each bus has a 32-bit edge-triggered register for data storage; data is stored on the rising edge of common clock input, CLK. An independent clock enable is provided for each register, so that new data can be clocked in or old data held; the clock enables are well-suited to a microprogrammed environment, and make the gating of clocks, always a risky business, unnecessary. The ability to clock or hold any register is a powerful tool for performing algorithms with conditional operations, or algorithms in which intermediate results must be delayed for one or more cycles before reentering the calculation.

In many applications,the internal registers will be used to store input and output operands; it is in this register-to-register mode that the chip shows its top speed. Some users, however, may wish to bypass one or more of the internal registers. The input and output registers can be made transparent independently using feedthrough controls FT0 and FT1. If all three registers are made transparent the device operates in a purely combinatorial "flow-through" mode. That mode, through, is somewhat slower than the register-to-register mode, but is useful in systems that need a register structure substantially different from that provided in the Am29325, or in systems where floating point operations must be concatenated with other combinatorial functions.

The two feedback data paths greatly simplify the task of moving data from one calculation to the next. One path routes data from the output of the arithmetic unit to a multiplexer at the input of register R; the multiplexer selects the operation result or R_0-R_{31}. The result of any operation can therefore be loaded into register

R, register F, or both. The second path feeds the output of register F to a multiplexer at the arithmetic unit's S port; the multiplexer selects either register S or register F as the port S input. This path effectively increases the number of commands—instruction R Plus S, for example, can also be performed as R Plus F.



4. Three programmable I/O bus modes permit the floating-point processor to operate with dual 32-bit input buses (a), a single, shared 32-bit input bus (b), or even two 16-bit buses (c) so that it can easily connect to most 16-bit microprocessor systems.

Thanks to the inclusion of three programmable I/O modes, the circuit readily interfaces with both 16- and 32-bit sytems. The most straightforward of these options is the 32-bit, two-input bus mode (Fig. 4a). The advantage of this mode is its high I/O bandwidth—no multiplexing of I/O buses is required, thus improving system speed and easing critical timing constraints. R and S operands are taken from their respective buses and clocked into the R and S registers on the rising edge of CLK; register F is also clocked on this transition.

Another choice sets up a 32-bit, single-input bus, in which both the R and S buses are connected to a single input bus (Fig. 4b). The R and S operands are multiplexed onto this bus by the host system; the R register clocks its operand on the rising edge of CLK, the S register on the falling edge. The S operand is double-buffered on chip, so that the new S operand is presented to the arithmetic unit on the rising edge of CLK. Operation of register F and the F bus is the same as in the 32-bit, two-input bus mode.

The last option has targeted 16-bit systems—a 16-bit, two-input bus mode (Fig. 4c). In this mode the R, S, and F buses are 16 bits wide; 32-bit operands are placed on the buses by time-multiplexing the 16 MSBs and LSBs of each data word. The LSBs of the R and S operands are double-buffered on chip, so that the complete 32-bit operands are presented to the arithmetic unit on the rising edge of CLK. Internal data paths and registers remain 32 bits wide, thus giving the 16-bit system designer the benefits of the simple interface and the speed of the wide internal data paths.

Putting the part through its paces

Multiplication and accumulation—a combination of operations very commonly used in digital filtering, image processing, matrix manipulation, and many other applications—can readily show the capability of the floating-point processor. In such a combination of operations, N input terms x_i are multiplied by constants k_i; the products are then added, producing the weighted sum:

$$s = \sum_{i=0}^{N-1} k_i \; x_i$$

To do this with the Am29325 is a simple two-step process, with two additional steps for initialization. In the first step data and coefficient values x_0 and k_0 are clocked into registers R and S. During step two the values x_0 and k_0 are multiplied and the product placed in register F; at the same time, data and coefficient values x_1 and k_1 are clocked into R and S. Third, values x_1 and k_1 are multiplied and the product placed in R. In step four, products x_1k_1 and x_0k_0 are added and the sum placed in F, and x_2 and k_2 are clocked into R and S.

The third and fourth steps are then repeated for as many iterations as needed to complete the operation. Once the part has been loaded with the first two sets of operands, the internal data path routes partial results to keep the arithmetic unit busy with a multiplication or addition every clock cycle; a new multiplication and accumulation is performed every two clock cycles. The partial results remain on-chip until the multiplication and accumulation is completed, thus eliminating I/O delays and the more complex programming that would result from having the adder and multiplier on separate chips.

Some real applications

A more specific application for the Am29325 could be its use as the computational engine in a fast Fourier transform (FFT) processor. During a FFT operation, word growth is incurred in the butterfly calculation, and if the FFT processor uses integer arithmetic, word growth can cause a system overflow. To prevent overflow, a scaling operation must be performed on the data. The overhead involved in checking for word growth overflow and scaling of data can be avoided by using floating-point arithmetic. Floating-point provides not only greater dynamic range but in most cases also provides greater precision (24 bits of significance versus 16 bits in a typical integer system).

A powerful, low-cost system that executes FFTs can be built around the floating-point processor (Fig. 5). It consists of a floating-point arithmetic processing unit, a data and coefficient address generator, a data and address storage block, high-speed data and coefficient memories, a system controller, clock generator, and host interface. Input operands to the R port are fed from the data store, while data to the S port is fed from the coefficient memory. The result of an arithmetic operation may be stored back in the data memory. An exclusive-OR gate is also available to complement the sign of the result, effectively multiplying the operand on the F bus by -1. For most operations, intermediate results can be held within temporary registers in the floating-point unit; only the final result need be sent off chip.

The high-speed data memory is made up of RAMs, the coefficient memory of PROMs. The data memory can be loaded with data from the host or can store results that have been processed through the floating-point chip. Once all data or results have been stored, the data memory is ready for use in an operation, or for transfer back to the host system. The coefficient PROMs contains the sine and cosine data required for an FFT, while the data store holds frequently used operands.

During the calculation of a butterfly, the same operands must be used in several different cycles—and since the data store reduces the number of memory read operations required, it speeds up data access. As the butterfly sequence progresses, the appropriate address is available from the address store, which consists of two more multilevel pipelined registers.

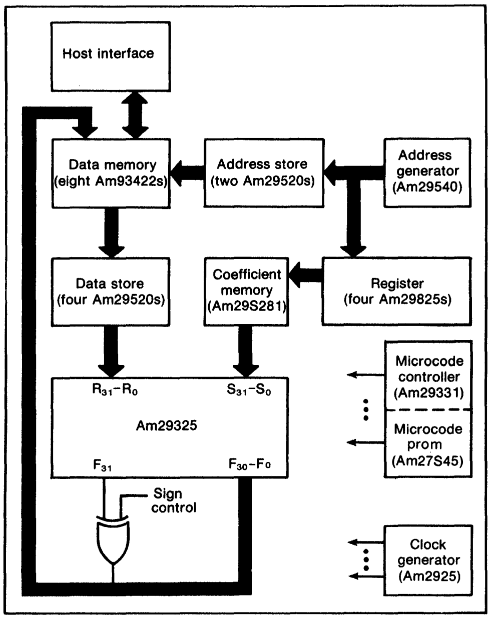
The host interface consists of a DMA channel that can perform high-speed block data transfers between the host system and the data memory. The system controller communicates with the host to receive or transfer data. It governs which operations are to be performed and how to perform them. Instructions are issued by the host computer, via the host interface, to the system controller, and the system controller informs the host when the operation is done.

The system controller consists of an Am29331 or similar microsequencer, and a microcode program stored in registered PROMs. The system clock generator uses an Am2925. The architecture allows a ten-cycle butterfly FFT to be executed (see Fig. 5 again) using a radix-2 decimation-in-time (DIT) algorithm. The equations for a radix-2 DIT algorithm are:

$$A' = A + BW\,B'$$
$$B' = A - BW, \text{ where all values are complex}$$

In cycles 1, 2, and 3, the first three operands are read from the data memory. Because of the overlapping butterflies, this read takes place while the previous butterfly is still being processed. In the following two cycles, data writes of the previous butterfly occur while the complex multiplications of (BW) are being performed. Cycle 6 reads in a new operand for the present butterfly and sums together the two products from the two previous cycles. In cycles 7 and 8, the real part of A′ and B′ is formed. In cycles 9 and 10, the real part of A′ and B′ is written to memory. Also, during these two cycles the other product pairs of (BW) are formed.

During cycles 11, 12, and 13, data for the next butterfly is read, and as part of cycles 12 and 13, the imaginary part of A′ and B′ is formed. In the following cycles the imaginary part of A′ and B′ is written to memory and processing of the next butterfly is initiated. The real and imaginary components of B′ have a negative sign, and can be corrected by complementing the sign. Counting the number of cycles from the first read or write of one butterfly to the next, it can be seen that a butterfly is computed every 10 cycles.


Host interface
Data memory (eight Am93422s)
Address store (two Am29520s)
Address generator (Am29540)
Data store (four Am29520s)
Coefficient memory (Am29S281)
Register (four Am29825s)
Microcode controller (Am29331)
Microcode prom (Am27S45)
R_{31}–R_0
S_{31}–S_0
Am29325
F_{31}
F_{30}–F_0
Sign control
Clock generator (Am2925)

5. To build a fast-Fourier transform processor that uses the floating-point processor as its heart requires only a few control chips and some memories. Use of the Am29540 and Am29332 LSI building blocks helps keep the circuitry simple.

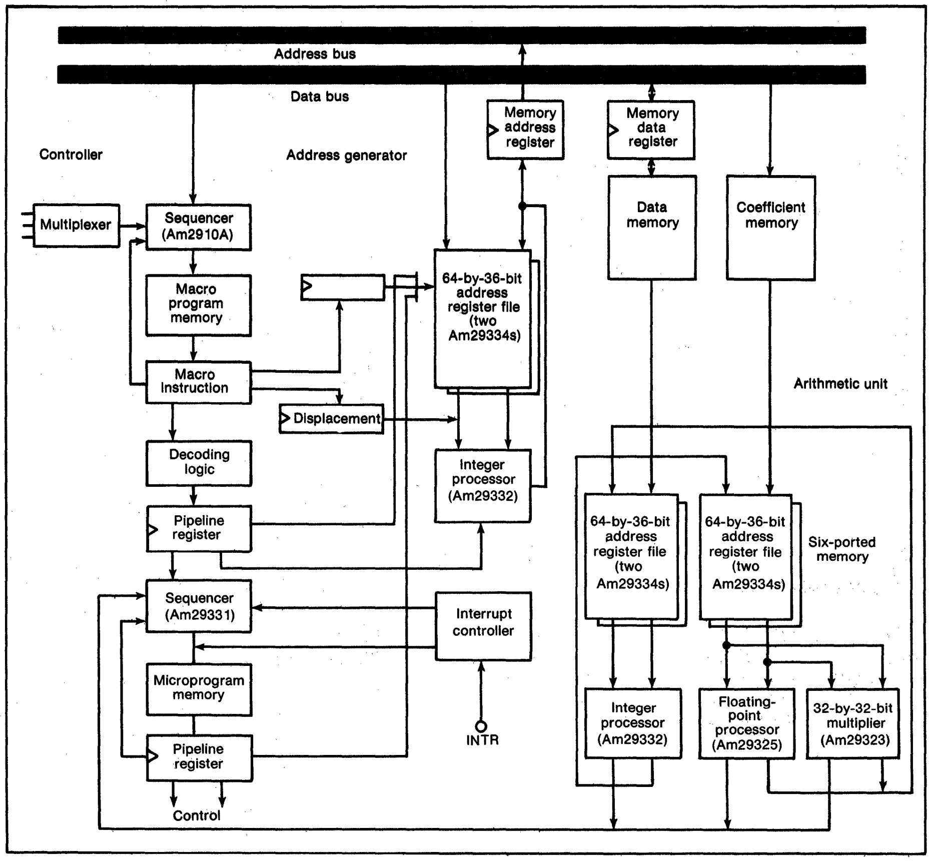
The big system picture

Although the floating-point chip fits well in small systems, it is also easily incorporated in larger, more powerful configurations. In one such system, a high-speed, microprogrammed integer and floating-point processor can be readily tailored to implement signal processing, image processing, or graphics algorithms (Fig. 6). The processor consists of a two-level controller, data and coefficient memory, address generator, and arithmetic unit. These functional blocks are considerably more flexible than their counterparts in the simpler FFT system.

The controller is divided into two levels, or sections: program and microprogram. In the topmost or program section, an Am2910A microprogram controller addresses a program memory that contains high-level instructions, or macros. These macros implement building-block operations; a graphics processor, for example, might have macros called Translate and Rotate that move objects in three-dimensional space. Each macro would carry with it parameters relevant to its operation, such as memory pointers or iteration count.

The program section passes address-related parameters to the address generator, and passes the iteration count and the decoded microinstruction start address to the microprogram section of the controller; this section then provides cycle-by-cycle control of processor resources during the execution of a macro. The heart of the micropogam section is an Am29331 microprogram controller—it addresses a microcode memory, in which the microprogram sequence for each macro type is stored.

The microprogram controller was chosen for three reasons: first, it can address up to 64 kwords, which makes possible a deep microprogram memory that can store many operation sequences. Second, its high speed permits the use of slower, less expensive microprogram memory, a particularly important considera-



6. A versatile, yet high-performance microprogrammable system can be built by including both the floating-point processor and the 32-bit multiplier into a system that uses the other Am29300 building blocks to form the control and address generation sections.

tion when the microprogram is large. And third, its micro-interrupt feature can be used to efficiently implement exception handling for arithmetic operations. By using interrupts for these exceptions, the overhead otherwise incurred in testing status flags can be greatly reduced.

The data and coefficient memories store input data, output data, and constants. In this application, data and coefficient memory have been separated from program memory. Sometimes referred to as a Harvard architecture, this approach increases throughput by allowing instruction fetch and operand fetch operations to proceed in parallel.

The address generator comprises a Am29332 ALU and two Am29334 register files. The register file stores up to sixty-four 32-bit base addresses and pointers. The Am29332 creates a 32-bit effective address from these bases and pointers, with the calculation assuming the forms:

base + pointer
base − pointer
base
or pointer

In addition, the Am29332 can perform mask, shift, and merge operations in a single cycle. This feature can be used to quickly calculate matrix addresses of the form:

$$a2^N + b,$$

where a and b are the row and column indices of the matrix element to be accessed. The combination of a 32-bit effective address and efficient matrix addressing makes this address generator particularly attractive for applications such as image processing, in which matrices must be plucked out of very large data arrays.

The arithmetic unit contains three arithmetic facilities—an Am29325 for floating-point operations, and the Am29332 and Am29323 for integer and logical operations. These devices accept data from a six-port register file made of four Am29334s. The register file has three purposes—it acts as a fast, temporary scratchpad for data, it routes data among arithmetic devices (the output of one arithmetic device can be written to the register file, and be used as an input operand by another such device during the following clock cycle), and it provides access to four data words every clock cycle, so that two or more arithmetic device can operate in parallel.

An example of this parallelism is integer multiplication-accumulation: because the Am29323 and the Am29332 receive operands independently, an integer product and sum can be calculated every clock cycle. The register file can then pass products from the Am29323 to the Am29332, for a throughput of one clock cycle per multiplication-accumulation.

Operation of the processor might be best understood by considering the execution of a typical macro. For graphics applications, one such macro is Translate, with which a set of points in three-dimensional space is moved in a given direction. The set of points is described by a list of vectors (X_i, Y_i, Z_i), while the translation is described by vector (S_T, Y_T, Z_T); each vector is stored in three contiguous data memory locations. Translation is performed by adding the translation vector to each entry in the vector list.

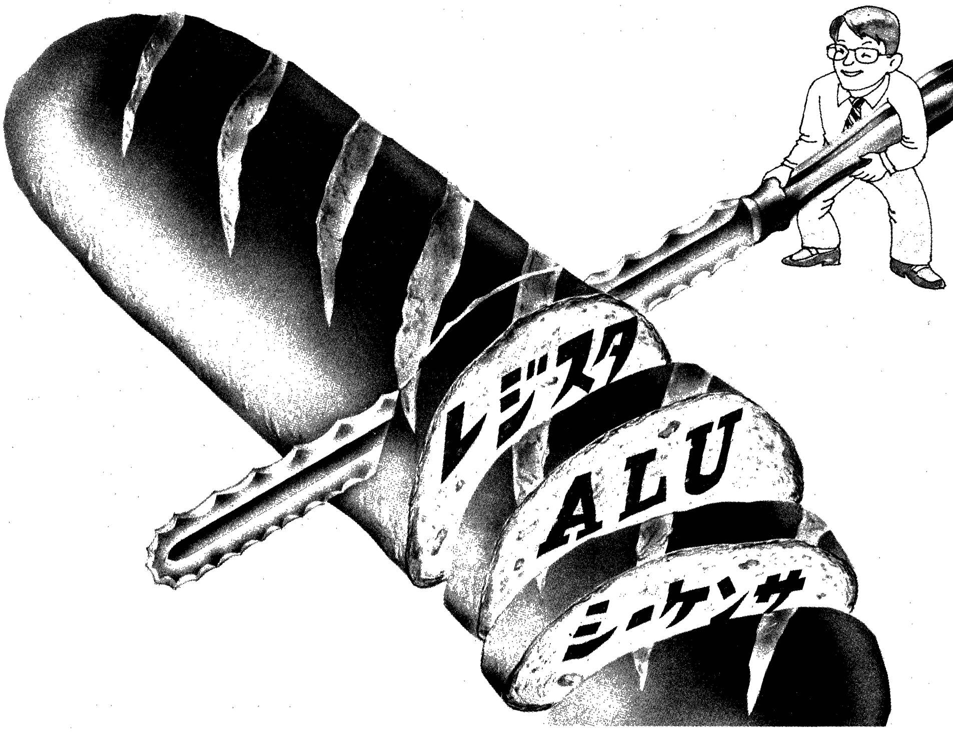
The translation process begins when the microprogram controllers encounters a Translate instruction in program memory. The Translate instruction is accompanied by three parameters: the start address of the translation vector, the start address of the vector list, and the number of vectors in the list. The first two parameters are passed to the address generator, the third to the iteration counter.

The microprogram section of the controller then assumes command, accessing the microcode for the Translate instruction. The microcode controls the address generator and arithmetic unit, specifying the operations needed to fetch each vector from the vector list, add the translation vector, and return the modified vector to the data memory. After all vectors in the list have been processed (as indicated by the iteration counter), control is returned to the Am2910A program sequencer, which then accesses the next macro from program memory.□

機能単位の分割法を採用した32ビット・バイポーラLSIファミリAm29300

ポール・チュー
バーナード・J・ニュー
米アドバンスト・マイクロ・デバイセズ社

米 Advanced Micro Devices, Inc. は，32 ビットでマイクロプログラマブルなプロセサ用のビルディング・ブロック・バイポーラ LSI ファミリ Am29300 の開発を進めている。シーケンサ，ALU，レジスタ・ファイル，並列乗算器，浮動小数点プロセサの 5 種の LSI がある。LSI の高集積化に対応して，Am2900 ファミリで採用したビット・スライス方式ではなく機能単位の分割法を採用した。内部 32 ビット構成とし，キャリ伝播の高速化を図っている。3 バス構成を採用し，データ転送バンド幅を広げた。パリティとマスタ/スレーブ機能を組み込み，耐故障性も実現できる。スーパミニコン，アレイ処理，ディジタル信号プロセサなど多様な応用分野での使用をねらっている。（本誌）



高性能なコンピュータ・システムの設計者は，32ビット並列処理のアーキテクチャを必要とする度合いを強めている。特に，インテリジェントな周辺端末やアレイ・プロセサ，ディジタル信号処理プロセサなどを設計する場合にその傾向が顕著である。しかしそれと同時に，32ビット・アーキテクチャで可能となる特徴，すなわち仮想記憶サポートやメモリ転送効率の向上，高い演算精度などをできるかぎり低コストに抑えて実現しなくてはならない。これまでは，特定用途向きに最適設計したVLSI回路を用いることが多かった。しかし，こうしたデバイスは柔軟性がないため，設計過程と設計した結果の両方があまり効率的でなかった。

Am29300ファミリは，汎用で32ビット，マイクロプログラマブルなビルディング・ブロック構成になっており，アーキテクチャ上の柔軟性が最高になるように設計してある。業界標準となったAm2900ファミリを基にしており，このファミリは特定用途の要求に合ったシステム・アーキテクチャを構築できる。また，フォールト・トレランスやフォールト検出のような多くの重要なシステムの課題に対応した基本機能も備えている。

Am29300ファミリは以下の基本回路から構成してある(図1)。

▷ **Am29331 マイクロプログラム・シーケンサ。** 16ビットの高速シーケンサであり，マイクロ命令レベルで割り込みとトラップを扱える。

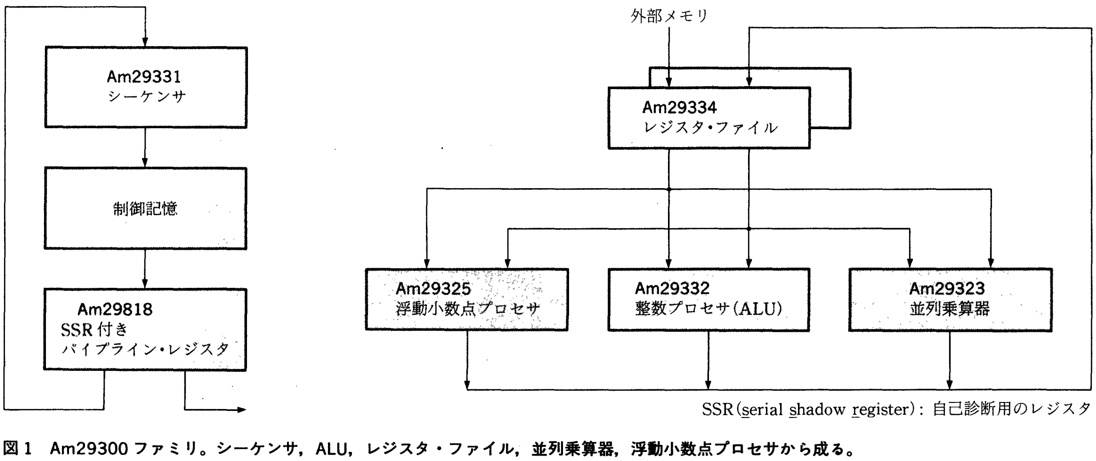
▷ **Am29332 ALU。** 32ビットの算術/論理演算とシフト操作を実行し，可変バイト長と可変ビット長のデータをサポートする。

▷ **Am29334 レジスタ・ファイル。** 64ワード×18ビット構成。デュアル・ポート構成になっており，リード動作とライト動作を同時に実行できる。

▷ **Am29323 並列乗算器。** 32ビット×32ビットの並列乗算器。外部に論理回路を付加することなしに，64ビット×64ビットと128ビット×128ビットにも拡張できる。

▷ **Am29325 浮動小数点プロセサ。** 単精度のIEEEかDECのフォーマットを用いる。1サイクルで加算と減算，乗算のほか，種々の変換操作を実行する。

このファミリは，システム・レベルでの進歩に呼応して，半導体コンポーネントのレベルにおいても機能が急速に複雑になっていくことを示している。顧客の要求がAm29300ファミリの市場を作り出したといえる。Am2900ファミリでは，レジスタ・ファイルとALUを1パッケージに収めてビット・スライスしていた。これに対しAm29300ファミリでは，ビット・スライスではなく，機能単位の分割を採用



Am29331 シーケンサ
制御記憶
Am29818 SSR付き パイプライン・レジスタ
外部メモリ
Am29334 レジスタ・ファイル
Am29325 浮動小数点プロセサ
Am29332 整数プロセサ(ALU)
Am29323 並列乗算器
SSR(serial shadow register): 自己診断用のレジスタ

図1 Am29300ファミリ。シーケンサ，ALU，レジスタ・ファイル，並列乗算器，浮動小数点プロセサから成る。

略語一覧(ABC順)

ALU:	arithmetic and logic unit
ARM:	availability, reliability and maintainability
ECL:	emitter-coupled logic
IEC:	International Electrotechnical Commission
IEEE:	The Institute of Electrical and Electronics Engineers, Inc.
MSB:	most significant bit
MSI:	medium scale integrated circuit
PROM:	programmable read only memory
RAM:	random access memory
SSR:	serial shadow register
TTL:	transistor-transistor logic
VLSI:	very large scale integrated circuit

して高い柔軟性を目指している。

機能単位の分割で柔軟性を上げる

機能単位の分割で得られる第一の利点はモジュラリティである。レジスタ・ファイルは他のデータ・パス・エレメントと分離してあるので，レジスタ・ファイル空間を容易に拡張できる。ある一定数のレジスタがALUチップに内蔵されている場合,デコーディングを容易にするためには,拡張に際してその一定数を単位として増加させる必要がある。このため，消費電力の問題を考慮すると，ALUに内蔵し得るレジスタ数は厳しく限定されることになる。これに対し，レジスタ・ファイルを独立させたAm29300ファミリでは，設計者は拡張時にレジスタの増加単位を自由に選べるようになる。また，電力を最大限に与えて必要な処理速度を引き出せる。

機能単位の分割によって，アーキテクチャ上の拡張も容易となる。レジスタとALUが結合していると，算術演算プロセサを付加する場合，レジスタの出力部に双方向ポートが必要になる。外部レジスタ内のオペランドがALUに転送できなければならず，内部レジスタ内のオペランドは算術演算プロセサに転送できなければならないからである。レジスタをALUから分離することによって，これらのポートは複数のソースとデスティネーションを持つ単方向バスでも良くなる。これも消費電力の節約になり，処理速度

〈資料請求番号 277〉

の向上につながる。

内部32ビット構成の採用で，キャリ伝播を高速化

ビット・スライス方式の主要な難点は，チップ間のキャリ伝播による時間の損失である。高速なルックアヘッド・キャリ方式を用いた場合でも，チップの外部を駆動するために生じる時間損失は非常に大きい。これを避けるために，Am29300ファミリの算術演算エレメントは完全な内部32ビット・データ・パスで構成してある。これに対しビット・スライスの場合は，カスケード接続に対するALUのキャリ伝播能力が限定されたものであるかぎり，拡張する場合は複数サイクルを使う拡張法に頼ることになる。ALU以外のデータ・パス・エレメントもすべてこの内部32ビット構成を採用している。

完全な32ビット・データ・パス・エレメントの第二の利点は，簡単にはスライスできないような機能を持たせられることである。このような機能の典型的な例としては，シフト・アレイと乗算器の二つが挙げられる。これらは両方とも，スライス間で転送するのが不可能なほど多くの情報をスライス間でやりとりしなくてはならない。優先度付けやマスク発生のような機能も，ビット・スライスが可能ではあるが，あまりきれいには拡張できない。Am29300ファミリでは，これらすべての機能がALUまたは32ビット乗算器のいずれかに用意してある。

3バス方式でデータ転送のバンド幅を広げる

この32ビット・データ・パスを持つデバイスの利点を十分に引き出して活用するためには，それに見合った広いバンド幅のデータ転送が要る。Am29300ファミリでは，3バス・アーキテクチャを用いることでこれを達成した。Am29334レジスタ・ファイルは2ポート構成ファイルで，各ポートからの同時アクセスができる。出力ラッチがあり，1クロック・サイクルでリードとライトを実行できる。各ポートには別々のリード用とライト用のアドレス入力があり，この動作の実現を容易にしている。

各データ・パス・エレメントには2組の32ビット入力ポートがある。これらを別々のオペランド・バスに接続でき

る。そして，これらのオペランド・バスにはレジスタ・ファイルが接続され,二つのオペランドが同時に載せられる。また，これらのデータ・パス・エレメントには結果を出力する 32 ビット・リザルト・ポートもある。このリザルト・ポートはリザルト・バスに接続し，これを経由してレジスタ・ファイルの一方の入力にデータを送り返す。この構成によって，3 アドレス方式のレジスタ-レジスタ間演算を 1 クロック・サイクルで完遂できる。

さらに広いデータ転送のバンド幅を得るためには，2 個のレジスタ・ファイルを用いてもよい。入力ポートを並列に接続し，データを二つのファイルに重複して書き込むことによって，単一のデータベースから 4 個のオペランドを同時にソースとして取り出すことができる(図 2)。二つの演算結果も同時にこのファイルに書き込める。この結果，2 組の演算エレメントが並列動作するのに十分なデータ転送能力が得られる。

この 3 バス・アーキテクチャの柔軟性のおかげで，これらのデバイスは他の種々の構成でも使える。信号処理やアレイ処理の応用では，乗算器と ALU を並列ではなく直列

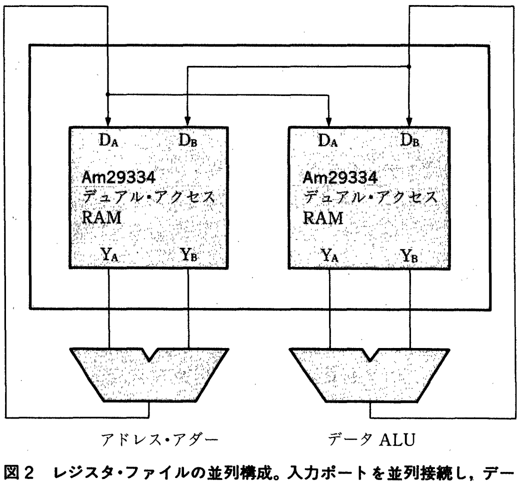


図 2　レジスタ・ファイルの並列構成。入力ポートを並列接続し，データを両方のレジスタ・ファイルに重複して書き込む。この結果，単一のデータ・ファイルから 4 個のオペランドを同時に読み出せ，データ転送のバンド幅を上げられる。

■製品についての定価、納期、カタログなどのお問い合わせは、最寄りの特約店へ
■技術的なお問い合わせは、横河・ヒューレット・パッカード株式会社・コンポーネント部(担当：稲垣、矢島、吉田)へ――〒168 東京都杉並区高井戸東3-29-21 TEL：03-331-6111 COMP506

〈資料請求番号 280〉

に配置できる。この結果，3個のオペランドに対する積和演算を最大限の速度で処理できる。

制御パスとデータ・パスの遅延をバランス良く改善

70〜80 nsのサイクル時間を達成するために，Am29300ファミリを開発するに際しては，いくつかの点を考慮して設計しなければならない。Am29300ファミリの各デバイスの内部をECL，入出力をTTLコンパチブルにすることとは別に，クリチカル・パス間でうまく処理速度のバランスを取る必要があろう。マイクロプログラム方式のシステムでは，サイクル時間は制御パスのエレメント（シーケンサとマイクロプログラム・メモリ）とデータ・パスのエレメント（レジスタ・ファイルとALU）によって定まる。制御パス，特にクリチカルな条件付き分岐に関するパスは，従来からボトルネックを形作っていた。

Am29300ファミリでは，従来の方法と比較して，制御パスとデータ・パスのタイミングを効果的にとっている。制御パスに関しては，Am29331がコンディション・コード・マルチプレクサやテスト条件発生，極性制御といった従来は外付けにしていた付属的な論理回路をチップ上に取り込んだ。また，このチップは処理速度を上げるのに最適な，改良したアーキテクチャを採っている。データ・パスに関しては，完全な32ビット単位の機能分割によって，キャリ・ルックアヘッド・パスの遅延のような相互接続遅延を排除した。

パリティとマスタ/スレーブを組み合わせて耐故障性を実現

システムのARMの概念（Availability, Reliability and Maintainability）は，設計者の間で大きな関心事になってきている。Am29300ファミリはパリティとマスタ/スレーブの二つの技術を組み合わせたオンチップの故障検出法を採用した。バイト単位のパリティが各データ・パス・エレメントのリザルト・バス上に発生され，これをAm29334レジスタ・ファイルに格納する。そして，データ・パス・エレメントのオペランド・バスに送出するときは常に再チェックする。こうしてデータ・バス上で生じる相互接続に起因した誤りを検出できる。偶数パリティを採用したので，

TTL バスのオープン状態も検出できる。オープン状態のTTL バスは高インピーダンス状態ですべて“1”となるが，これはエラー状態と同じだからである。

機能の検証に対しては，動作のマスタ/スレーブ状態を利用して2ユニットを並列に接続すればよい(図3)。一方のユニットが実際に計算処理を実行し，もう一方が各サイクルごとにその結果をチェックする。したがって，スレーブ・ユニットがマスタの正常な動作を検証することになる。さらに，マスタ・ユニットはその内部処理結果と出力バス上のデータを比較チェックし，他のデバイスが必要のないときに外部バスを駆動していないことを確認する。検出された誤りはすべてマイクロ命令レベルで割り込みを起動できる。従来の冗長構成と異なり，専用ソフトウェアは必要ない。冗長機能ユニット間の通信に起因するシステム性能の低下も起こらない。パリティ・チェックとマスタ/スレーブ動作のこの組み合わせと，高価なソフトウェアでなくコスト面で有利なハードウェアを使用することは，将来の冗長システムを設計するうえでの鍵となる。

この機能検証用のマスタ/スレーブ構成は，さらにシステム・レベルに拡張できる。二つの同一の機能ユニットを持ち，各ユニットが，それぞれ別々のボード上でマスタ/スレーブ構成になっているシステムを考えてみよう。正常動作

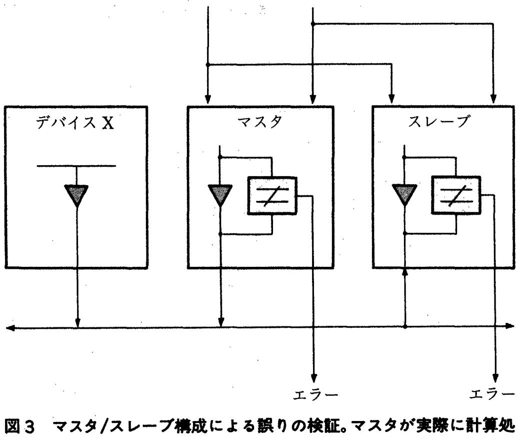


図3 マスタ/スレーブ構成による誤りの検証。マスタが実際に計算処理を実行し，スレーブが各サイクルごとにその結果をチェックする。

している間は，異なったタスクを各機能ユニット上でスケジュールし，実行できる。しかし，誤りを検出した場合には，誤りを起こしたボードで実行中のタスクは，バックアップとなるもう一方の機能ユニット上で再実行させる。言い換えれば，この耐故障構成方式では，4倍のハードウェア量でほぼ2倍近い性能を達成できるのである。これに対し，従来の典型的な多数決構成を採るシステムでは，3倍のハードウェア量を投入して性能はそのままである。

複雑なパイプライン構成を避ける

アーキテクチャ上の柔軟性を損なわずに，最大限の性能を引き出すということは，言外にパイプライン化するということを意味している。段数の多いパイプライン制御は，反復性の高い特定の操作に対しては有効であろう。しかし，システムの無駄な待ち時間が増えることに加え，より多くのソフトウェアとハードウェアが必要になる。この無駄な待ち時間は，ある計算が直前の演算結果に依存するような場合，たとえば再帰的なアルゴリズムを実行するようなときに大きな問題となる。このような場合にはパイプライン動作に待ち時間が生じ，この遅れがスループットを下げるためである。

レジスタ・アドレスの事前選択機能を実現するためのかなりの量の付加的ハードウェアも別の遅延をもたらす。これもパイプライン化を望まないユーザにとっては不利益となる。パイプライン化は分岐動作時にも遅延が出るし，マイクロプログラムの記述と理解も難しい。これらの理由から，Am29300ファミリは単一レベルのパイプライン・アーキテクチャを採用した。

高性能，小型，低コストの多様なシステムを実現できる

半導体コンポーネントの最終的な評価は，目的とする応用からの要求をそのコンポーネントがどれだけよくサポートしているかで決まる。汎用のスーパミニコンとディジタル信号プロセサでは，その機能と性能に対する要求が大きく異なっている。しかし，機能単位の分割と単純な3バス・アーキテクチャによって，Am29300ファミリのデバイスは多様な応用分野からの要求を満たすのに適している。

〈資料請求番号 **282**〉

図4はAm29300ファミリのコンポーネントを用いて構成したマイクロプログラム方式のスーパミニコンの例である。データ・パスはAm29332 ALU(プロセサ)，アクセラレータとしてのAm29323並列乗算器，およびAm29334レジスタ・ファイルから構成してある。この構成では，アドレスの計算とデータ演算は順次実行する。この構成とは違って，Am29334を並列に置き，効率良く6ポートのレジスタ・ファイルを実現することができる。こうすると，1マイクロサイクルごとに四つのリード・アクセスと二つのライト・アクセスを実行できる。この構成を採ると，もう一つのAm29332を，通常のALU動作と並列にアドレス計算する専用ALUとすることが可能になる。レジスタ空間は両者のALUで共有する。60〜80 nsのマイクロサイクル時間を生かして，標準的なスーパミニコンの数倍の性能を持つ

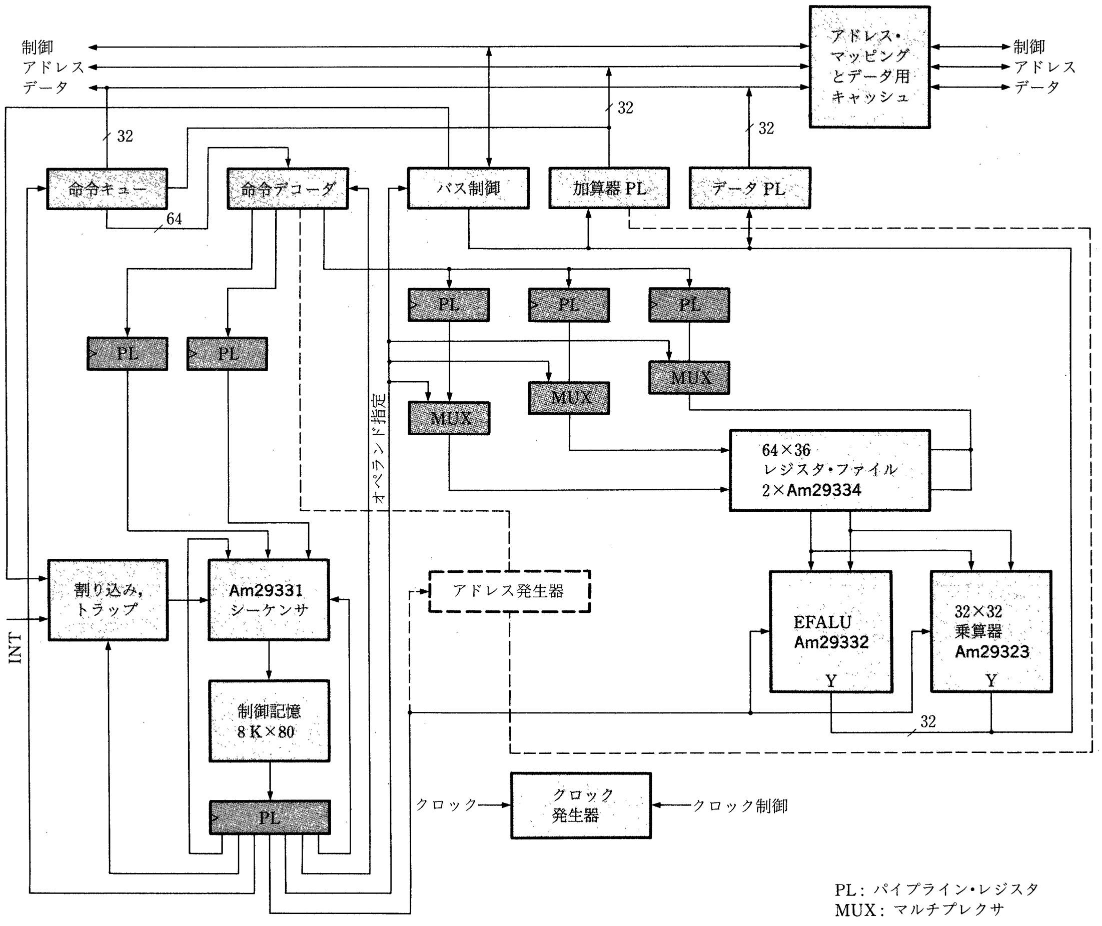


図4 Am29300ファミリを用いたスーパミニコンの構成例。データ・パスはALU，並列乗算器，レジスタ・ファイルから構成してある。この例ではアドレス計算とデータ演算は同一ALUで順次実行する。

プロセサ/コントローラ・サブシステムを，Am29300 ファミリのチップを用いて構成できる。そのうえ，このサブシステムは標準的なスーパミニコンに比べてボードの実装に必要な空間がはるかに小さくてすみ，消費電力も格段に小さくなろう。

3 バス・アーキテクチャは多くのアレイ・プロセサで使っているものと非常によく似ている。この構成はバス・バンド幅が広いので，演算ユニットを最大限に動作させられる。さらに高いバンド幅が必要なら，前述のようにレジスタ・ファイルを並列接続して 6 ポート・メモリとすることで，2 組の演算エレメントを並列動作させられる。この構成のバス・バンド幅で動作させる浮動小数点アレイ・プロセサは非常に強力なものとなろう。実効的に 20 MHz のスループットというのも不合理なことではない。

機能単位の分割は結果的に VLSI ビルディング・ブロックとなるが，MSI 並みの機能的柔軟性も併せ持つ。従来はこれを MSI コンポーネントで作っていた。しかし，これらの各ブロックの演算処理能力は VLSI の複雑性を反映して大きい。こうして，他の方法では膨大な回路規模を必要とするために不可能であるような専用システムを実現できる。

この処理能力と柔軟性の組み合わせによって，設計者は高性能のスーパミニコンやアレイ・プロセサなどを実現できる。しかも，これらは小型でかつコスト面でも優れている。さらに，Am29300 ファミリは，マイクロプログラム方式の利点を生かせるなら，いかなる汎用の応用にも適用できよう。

最大 64 K ワードの制御記憶をアクセスする——Am29331 マイクロプログラム・シーケンサ

Am29331 は 16 ビットのマイクロプログラム・シーケンサで，最大 64 K ワードの制御記憶をアドレスできる。このチップの最も重要な設計課題は処理速度であった。このため，従来は外付けで一般にシステムのクリチカル・パスになるコンディション・コード・マルチプレクサとテスト論理発生器，極性制御論理をチップ上に集積した。

図 5 は Am29331 マイクロプログラム・シーケンサのブ

ロック図である。Dバスには通常の分岐アドレスとパイプライン・レジスタからのワード・カウント値が乗せられる。Aバスには Dバスとは違うもう一つのアドレス・ソースが乗せられる。多くの場合はマッピング用の PROM で，さまざまなマイクロプログラム・ルーチンの開始アドレスが格納してある。

4組の4ビット・マルチウエイ入力があり，同時に起こる複数の外部テスト条件によって，シーケンサは16方向に分岐できる。アドレス・レジスタとインクリメンタの組み合わせは，連続したアドレス発生のためにプログラム・カウンタの機能を果たす。カウンタはループ数をカウントするためにワード・カウント値を記憶する。スタックはサブルーチン・リターン・アドレスや割り込み/トラップ・リターン・アドレス，ループ・アドレス，ループ・カウントを格納する。33 レベルあり，サブルーチンとループ，割り込みのネスティングができる。スタックおよびスタック・ポインタの内容も双方向のDバスを介してアクセスできる。アドレス比較レジスタでは，前もって格納してあるマイクロプログラム・アドレスを動的にモニタし，ブレークポイントの生成やプログラム統計値の収集ができる。

Am29331 のもう一つの特徴は，マイクロ命令の境目で割り込みやトラップを扱える点である。割り込みとトラップは外部からの各種要求に対し最小の遅延時間で応答するために，よく使う二つのテクニックである。割り込みとトラ

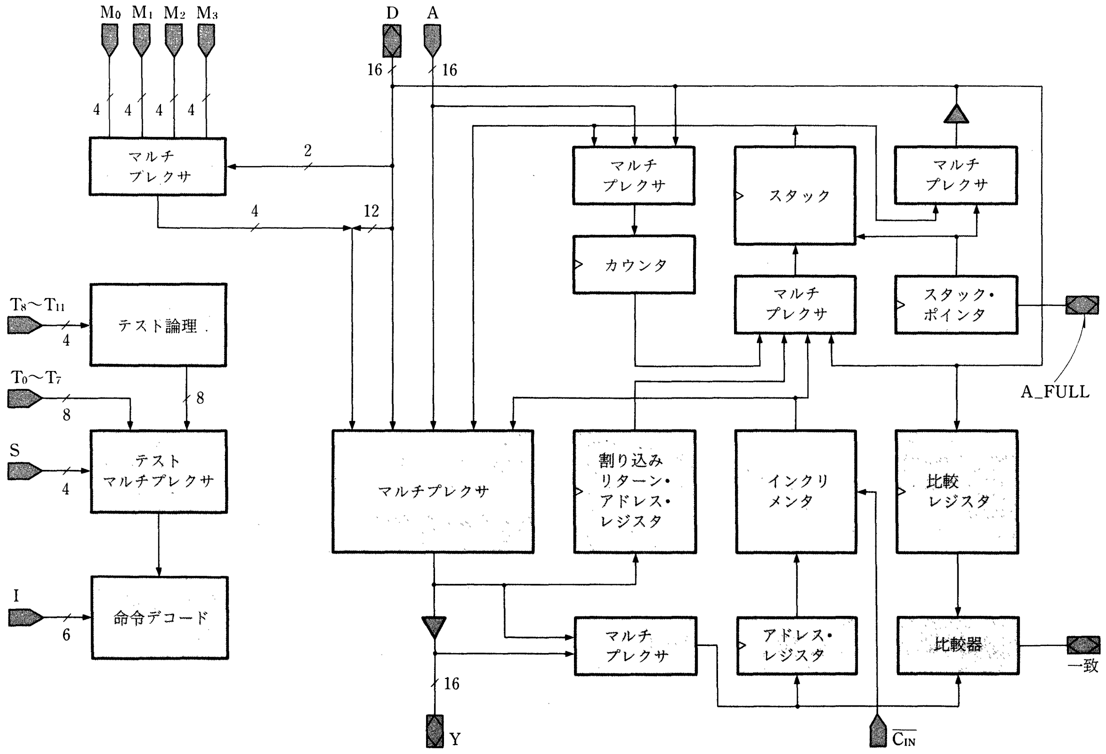


図5 Am29331 シーケンサのブロック図。最大 64 K ワードの制御記憶をアドレスできる。コンディション・コード・マルチプレクサとテスト論理発生器，極性制御論理回路を内蔵させ，処理速度の向上を図った。

ップのどちらを選ぶかは，設計者の趣味の問題である。Am29331はこのどちらも扱える。

割り込みは予期しないプロシジャ・コールとして取り扱う。割り込みがかかると，シーケンサの出力であるYバスはオフになり，外部の割り込みベクタが制御記憶へのアドレスを供給できるようになっている。同時に，割り込みベクタ・アドレスに1を加えた値をプログラム・カウンタに書き込む。割り込みリターン・アドレスはスタックに退避し，割り込みのネスティングができる。

これに対し，トラップは実行中のマイクロ命令を必ず中断しなければならない条件である。トラップが起こると，実行中のマイクロ命令はアボートされる。シーケンサの出力であるYバスはオフされ，トラップ・ベクタによってトラップ・ルーチンに飛び，しかるべき対応手段を取る。中断したマイクロ命令のアドレスは後で再実行するためにスタックに押し込まれる。

Am29331マイクロプログラム・シーケンサの命令セットは64種の命令から成る。命令セットの構造は高級言語に似ており，構造化マイクロプログラミングを容易に行なえる。

ビット・フィールド操作もサポート――Am29332 ALU

Am29332は32ビットのALUで，汎用の数値演算と可変長ビット・フィールド操作をサポートする強力なプリミティブを備えている。2組の入力バス(DA，DB)と1組の出力バス（DY）から成る3バス・アーキテクチャになっている。入力バス（DA，DB）にはソース・オペランドだけでなく，バイト単位のパリティも入力される。パリティは相互接続に起因する誤りを検出するためにチェックされる。同様に，出力バス(DY)には演算結果だけでなく演算直後に生成されるパリティも出力する。

Am29332には種々の機能ブロックがある（**図6**)。ALUは3組の32ビット入力を持つ。M入力はマスク発生器に接続しており，他の二つのオペランドの演算すべき部分を選択する。通常の算術論理演算では，マスクは可変バイト長のオペランド（最大4バイト）を選ぶ。ALUのステータス・ビット（負，オーバフロー，ゼロ，キャリ）は適切なバイト境界で発生する。S入力の選択されなかったバイトは

そのままALUを通過する。

フィールド論理演算の場合は，マスクは演算に際して可変長ビット・フィールドのオペランドを選択する連続したフェンスの役目を果たす。マスクは外部からの5ビット入力か内部レジスタの値によって発生する。マスクは目的の位置に合わせるためにシフトすることもできる。どれだけシフトさせるかは，外部からの6ビット入力か内部レジスタの値で与える。

ファネル・シフタは，64ビット入力/32ビット出力のアップシフタなのでそう呼ばれるが，32ビット・バレル・シフト，Nビットの算術/論理アップ・ダウン・シフトを実行できる。ファネル・シフタとマスク発生器の組み合わせでワード境界にまたがったオブジェクトの抽出と整列をサポートできる。これはグラフィックの応用で有効な操作である。

プライオリティ・エンコーダはMSBのビット位置を示す2進コードを出力する。このプライオリティ・エンコーダとファネル・シフタの組み合わせによって，浮動小数点（データ）の仮数を2サイクルで正規化できる。ALU出力

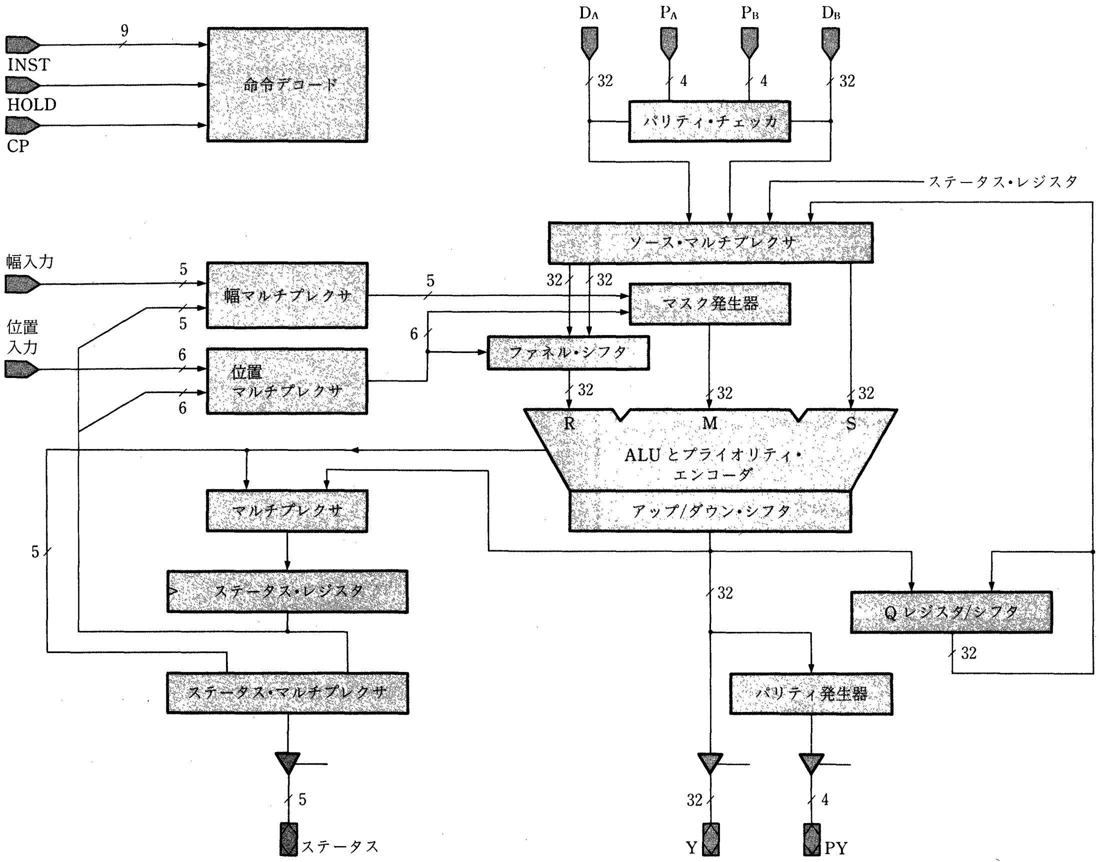


図6 Am29332 ALUのブロック図。2組の入力バスと1組の出力バスから成る3バス・アーキテクチャになっている。汎用の数値演算と可変長ビット・フィールド操作をサポートする。

部のシフタとQシフタ/レジスタは主に乗算と除算に使う。変形したBoothのアルゴリズムを用いた2ビット同時の乗算と，すべての符号の組み合わせに対応した引き放し法の除算をサポートしており，演算の最後に実行する，符号の組み合わせに対する修正ステップも組み込んである。32ビットのステータス・レジスタはAm29332のコンテクストを記憶しており，マイクロ割り込み時には退避と再格納を各々Yバスと DA/DB バスを介して行なえる。

命令セットは128種から成り，対称性と直交性がある。このおかげで，コンパイラでコードを生成するのが容易になっている。ソース・オペランドの順番によって結果が違う操作(たとえばB−AとA−B)は両方を用意してある。さらに，すべての算術論理演算はデータ・タイプと完全に独立している。したがって，同じ命令を1，2，3，または4バイトのオペランドに対して使える。

完全2ポート構成を採用——
Am29334 レジスタ・ファイル

Am29334は完全な2ポート・レジスタ・ファイルである。バイト単位のパリティを記憶できるように64ワード×18ビット構成になっている(図7)。完全に独立した2個のポートがあり，どちらのポートも入力バスと出力バス，リード制御とライト制御が別々にある。二つの独立なリードまたはライト・アクセスが同時にできる。このため，デュアル・リードやオーバラップしたリードとライトが可能である。デュアル・ライトもサポートしているが，同一のアドレス・ロケーションに対するデュアル・ライトだけは除いている。この場合はデータ誤りを発生させるからである。

各ポートにはアドレス入力も2組ある。一方はリード用で他方はライト用である。ライト動作中にリード・アドレスを設定したり，その逆もできる。リード/ライト・アドレス・マルチプレクサを内蔵したので，入出力遅延が1段分少なくなり，サイクル時間が改善された。タイミング制御も簡単になった。ライト・エネーブルとマルチプレクサ選択のタイミングは両方ともWE入力信号から内部で生成する。バイト単位の書き込みもできる。カスケード接続すれば，バイト，ハーフ・ワード，フル・ワードのいずれか

でアップデートできる。

32ビット乗算を1サイクル，64ビット乗算を4サイクルで実行――Am29323 32ビット乗算器

Am29323は32ビット乗算器であり（図8），単一クロック・サイクルで64ビット長の積が得られる。完全なBoothのアルゴリズムを用いて構成したアレイ乗算器を内蔵している。オペランド・レジスタとリザルト・レジスタがあり，64ビット長の出力はレジスタに格納してからマルチプレクスして32ビット出力ポートに時分割で出力する。オペランド・レジスタとリザルト・レジスタは各々独立にデータをパスさせることができる。このおかげで，このチップはさまざまなパイプライン構成に適用できる。小数点以下の2の補数演算の際に，結果の桁を入力と合わせることができるように，出力部に1ビット・シフトの機能が備わっている。

Am29323は高精度演算も扱う。乗算器アレイの出力部にあるアキュムレータによって，部分積の整列と加算ができる。倍精度入力レジスタは，複数サイクルにわたる64ビット乗算の間，オペランドを格納しておくことができる。4個の部分積が形作られて合算される。テンポラリ・レジスタ

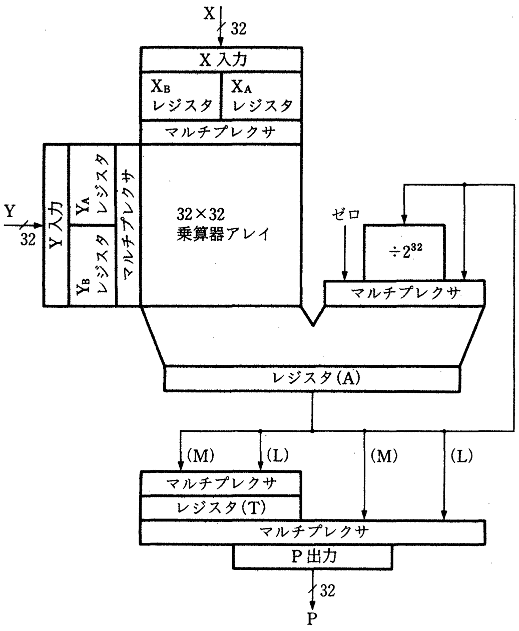


図8 Am29323 32ビット乗算器。単一クロック・サイクルで64ビット長の積が得られる。Boothのアルゴリズムを使用。64ビット長の出力は時分割で32ビット出力ポートに出す。

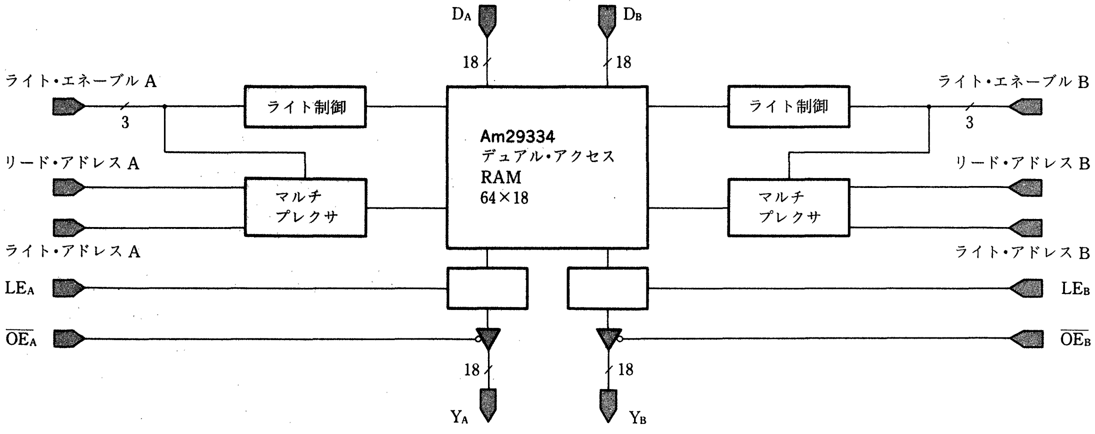


図7 Am29334 レジスタ・ファイル。64ワード×18ビット構成になっており，バイト単位のパリティを記憶できる。完全な2ポート構成になっており，どちらのポートも入力バスと出力バス，リード制御とライト制御を別々に持っている。

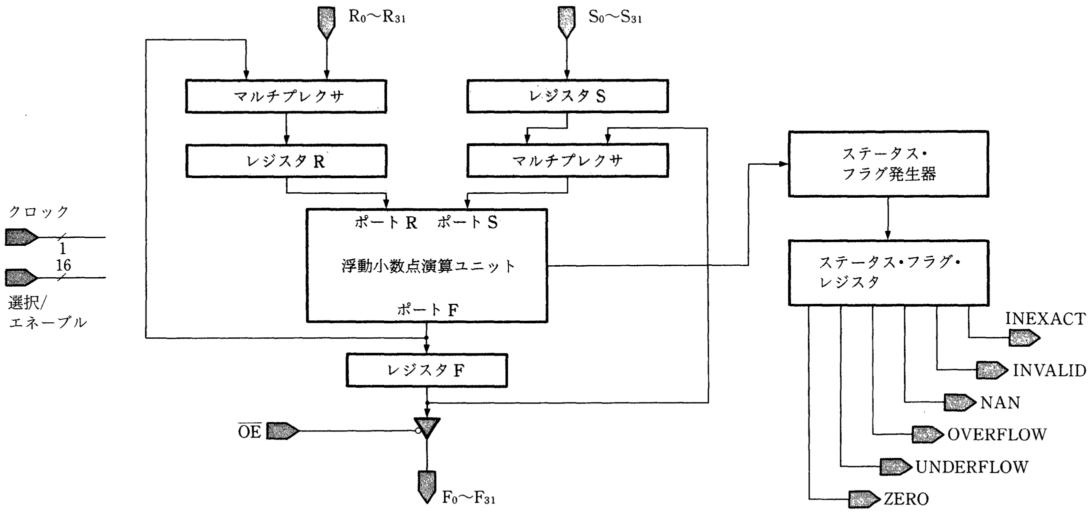


図9 Am29325浮動小数点プロセサのブロック図。加算，減算，乗算を実行する。除算に対するサポート機能もある。各演算は1サイクルで実行し，IEEEまたはDECフォーマットのいずれかを用いる。

は出力を転送する際のスケジューリングを補佐するためにある。パイプライン化した実効的なスループットとしては，64ビット乗算を4サイクルごとに1回実行できる。

アキュムレータは96ビットと128ビットの乗算をサポートするのに十分な幅がある。しかし，これらの演算に対しては，入力データを繰り返し与えることが必要になる。

ファミリ外での一般的使用も重視——Am29325浮動小数点プロセサ

Am29325浮動小数点プロセサは単精度浮動小数点演算のためのアクセラレータとして働く。サポートしている算術演算は，加算と減算，乗算であるが，除算に対するサポートもできる。これらの演算は1サイクルで実行し，IEEE(IEC)またはDECフォーマットのいずれかを用いる。この両フォーマット間の変換機能や固定小数点と浮動小数点との間の変換機能を持つ。

Am29300ファミリのメンバではあるが，Am29325は高速浮動小数点エンジンとして一般にも使えるように設計した。32ビットの3バス入出力構成（**図9**）であるが，16ビットの1バス・アーキテクチャと互換なモードでも動作できる。入力と出力レジスタはそのままパスさせることもでき，多様なパイプライン構成に対応できる。たとえば，入力と出力の両方のレジスタをそのままパスさせるようにすれば，完全な組み合わせ回路の演算となる。

この柔軟性のおかげで，このチップは信号処理システムやアレイ処理システムでの応用に最適である。こうしたシステムでは最も頻繁に行なう演算は積和演算であり，これによって，一般に畳み込み，相関，マトリクスなどの演算を容易に実行できる。

この積和演算をサポートするのに，Am29325では二つの方法がある。一つは結果を出力レジスタではなく入力レジスタに格納し，結果がただちにオペランドとして使えるようにする方法である。もう一方は，出力レジスタをオペランドとして使う方法である。この方法では，二つのオペランドの積をとり，その積を次の命令のオペランドにできる。この次の命令は積を出力レジスタに加える。出力レジスタはアキュムレータとして働く。積和演算は最低限の2サイクルで完了する。●

P. Stuhlmüller,
U. Renz

Echtzeit-Fouriertransformation ist eine bei der Analyse von Analogsignalen immer wieder benötigte Funktion. AMD's LSI-Familie Am29500 öffnet den Weg zu außerordentlich schnellen Hardware-Lösungen.

Digitale Signalverarbeitung mit bipolaren LSI-Bausteinen

Der Entwurf von Hochleistungs-Prozessoren für die digitale Echtzeit-Signalverarbeitung erfordert schnelle mikroprogrammierbare Bausteine, die den außerordentlich hohen Datendurchsatzraten in diesem Anwendungsbereich gerecht werden. Die Bit-Slice-Familie Am2900 scheint hierfür prädestiniert [2], ist jedoch nicht optimal, da für digitale Signalverarbeitung umfangreiche, sehr komplexe Software zu erstellen ist. Speziell für diesen schwierigen Anwendungsbereich und im besonderen für die schnelle Fourier-Transformation (FFT) hat *Advanced Micro Devices* mit der Serie Am29500 eine neue Bausteinfamilie entwickelt. In einem sehr schnellen bipolaren LSI-Prozeß gefertigt, bieten diese ICs den für komplexe Echtzeitfunktionen notwendigen enormen Datendurchsatz.

Zunächst sind vorerst vier Bausteine der ersten Generation im Design abgeschlossen.

- ▷ Am29502: mikroprogrammierbarer Signal-Prozessor mit registerorientierter ALU in Byte-Slice-(8 bit)-Architektur
- ▷ Am29516/17: Parallel-Multiplizierer mit 16×16 bit und 50 ns Produkt-Zykluszeit
- ▷ Am29520/21: Pipeline Register mit 4 Ebenen in Byte-Slice-Architektur mit Multiplexer
- ▷ Am29540: Einchip FFT-Adreß-Generator für vielfältige FFT-Algorithmen.

Die spezifischen Eigenschaften der jeweiligen Bausteine sind basierend auf der Philosophie der Familie Am2900 für den Bereich digitaler Signalverarbeitung zugeschnitten. Diese Philosophie ergibt ein Maximum an Datendurchsatz, verbunden mit der Flexibilität der Mikroprogrammierung von Bit-Slice-Konfigurationen. Die Besonderheit der Familie Am29500 ist jedoch, daß bereits wenige Bauelemente mit einem Minimum zusätzlicher SSI-Bausteine eine vollständige Systemlösung ermöglichen.

Vergleicht man Anwendungen in der digitalen Signalverarbeitung mit anderen in der Prozeßrechentechnik, sind insbesondere die häufigen Multiplikationen von Abtastwerten zu nennen, die praktisch in allen gebräuchlichen Algorithmen auftreten. Es ist deshalb keinesfalls überraschend, daß die erste Generation der Familie Am29500 einen ultraschnellen Hardware-Multiplizierer enthält.

Dipl.-Ing. Peter Stuhlmüller ist Spezialist für Datenverarbeitung und Mikroprozessorenanwendungen in München, **Udo Renz** ist Applikationsspezialist bei *Advanced Micro Devices GmbH.*

Der 16 bit Parallel-Multiplizierer Am29516/17

Die als zeitintensiv bekannte Multiplikation ist eine Funktion, deren Erweiterbarung in der Wortlänge bei Bit-Slice-Systemen nicht ohne Hindernisse ausführbar ist. Um dem umfangreicheren Bausteinaufwand bei der ohnehin langsamen Softwarelösung für Operanden mit 16 bit aus dem Weg zu gehen, wurde die zeitlich nicht zu übertreffende parallele Hardwarestruktur gewählt und für die feste Wortbreite von 16 bit ausgelegt. Der nach diesem Prinzip bereits verfügbare Baustein MPY-16HJ von *TRW* [3] paßt jedoch nicht in die Philosophie der neuen Familie Am29500.

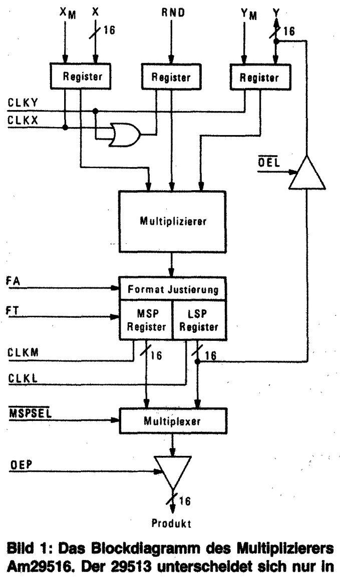


Bild 1: Das Blockdiagramm des Multiplizierers Am29516. Der 29513 unterscheidet sich nur in der Ansteuerung der Register

Deshalb wurden zwei noch leistungsfähigere Versionen geschaffen, die optimierte Verknüpfungsmöglichkeiten zu den anderen Systembausteinen aufweisen.

Der MPY-16HJ besitzt eine feste Rückkopplung des LSP-Ausgangs mit dem Y-Eingang, die in gewissen Fällen jedoch nicht brauchbar ist. In busorientierten Systemen, die nach dem linearen Pipeline-Prinzip konzipiert sind, würden hier Probleme im Datenfluß auftreten. Die Lösung in *AMD's* Baustein ist ein angefügter Produkt-Multiplexer am Ausgang der MSP- und LSP (Least Significant Product)-Register und eine Steuerbarkeit der Rückkopplung vom LSP-Register auf den Y-Eingang durch einen zusätzlichen Signaleingang $\overline{OEL}$. Hat dieser niedrigen Pegel, liegt das LSP am Y-Eingang an, bei Umschalten auf hohen Pegel liegt Tri-State-Zustand vor. Mit dem Multiplexersteuereingang $\overline{MSPSEL}$ kann das LSP-Register wahlweise auf Rückkopplung oder auf den Produktausgang geschalten werden. Damit sind Am29516 und 17 einerseits voll kompatibel zu den TRW-Bausteinen, vermeiden jedoch deren genannten Nachteil.

Trotzdem wurde damit die Erfordernis für mikroprogrammierte Systeme noch nicht vollständig erfüllt. Um dies zu erreichen, müssen die Ein- und Ausgangs-Register durch den Mikrocode mit vier unabhängigen Taktzyklen steuerbar sein. Dies wäre nicht ohne zusätzliche SSI-Bausteine möglich. Die Freigabe auf diese Weise zu erzeugen, bedingt eine Anpassung der Flanken. Um dies zu umgehen, wurden beim Am29517 die Registersteuerungen durch einen Einzeltakt und drei Register-Freigabeeingänge (X-Register, Y-Register und Produkt-Register) ersetzt (Produkt-Register ≙ 32 bit). Das Blockdiagramm der Multipliziererbausteine Am29516 und 17 ist in **Bild 1** dargestellt.

Die Multiplizierer stellen ein maßgebendes Element eines digitalen Signalverarbeitungssystems dar. Weitere wichtige Bausteineigenschaften sind Speicherwortbreite, ALU-Kapazität (Leistung des Rechenwerks) und einfache Zusammenschaltbarkeit zur Hardware eines Signalprozessorsystems. Die Zusammenschaltung der Multiplizierer Am29516 und 17 mit Registern, ALUs oder anderen Komponenten der Familie AmAm2900 ist nicht ohne weiteres möglich und normalerweise auch nicht vorgesehen. Die Mikroprogram- ▷

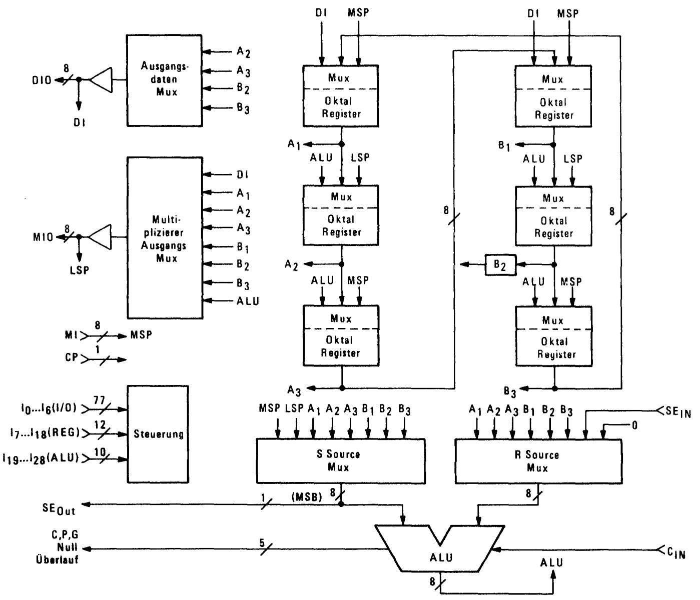


Bild 2: Aufbau des Byte-Slice-Prozessors Am29501

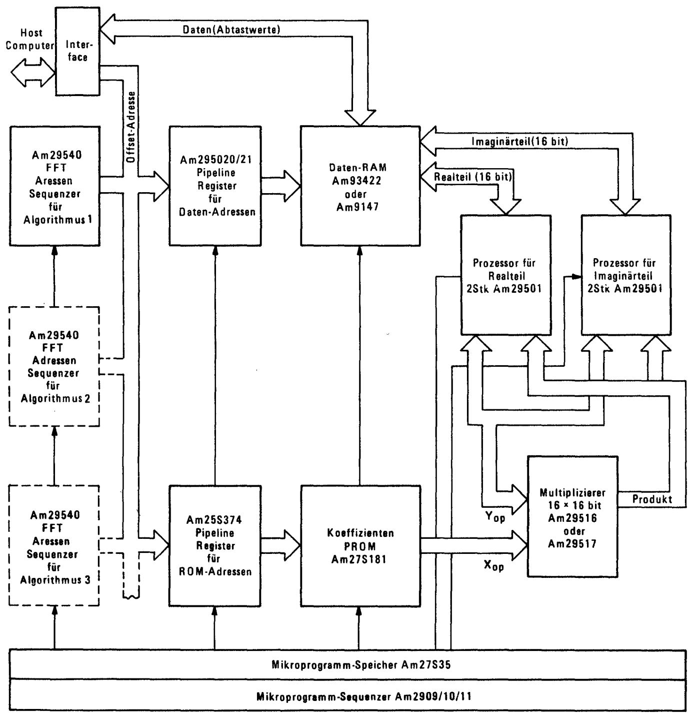

Host Computer
Inter-face
Daten(Abtastwerte)
Offset-Adresse
Am29540 FFT Aressen Sequenzer für Algorithmus 1
Am295020/21 Pipeline Register für Daten-Adressen
Daten-RAM Am93422 oder Am9147
Imaginärteil(16 bit)
Realteil (16 bit)
Prozessor für Realteil 2Stk Am29501
Prozessor für Imaginärteil 2Stk Am29501
Am29540 FFT Adressen Sequenzer für Algorithmus 2
Am29540 FFT Aressen Sequenzer für Algorithmus 3
Am25S374 Pipeline Register für ROM-Adressen
Koeffizienten PROM Am27S181
Yop
Xop
Multiplizierer 16 × 16 bit Am29516 oder Am29517
Produkt
Mikroprogramm-Speicher Am27S35
Mikroprogramm-Sequenzer Am2909/10/11

Bild 3: Grundkonfiguration eines FFT-Prozessors mit je zwei Byte-Slices Am29501 für Real- und Imaginärteil (je 16 bit)

mierung würde außerordentlich aufwendig sein, insbesondere wenn konjugiert komplexe Zahlen berechnet werden müssen.

Der Prozessor Am29501

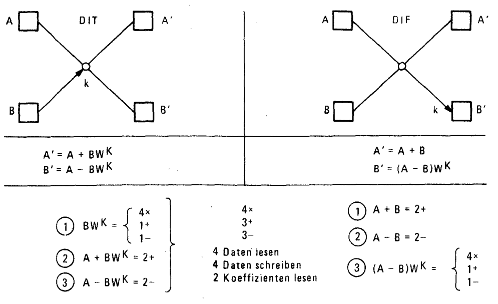
Speziell für die digitale Signalverarbeitung wurde als universeller mikroprogrammierbarer Prozessor der Am29501 konzipiert. Die Register/ALU-Struktur ist in diesem Baustein optimal verwirklicht und in Byte-Slice-Format ausgeführt. Die Anordnung der internen Elemente ist in mehrfach redundanten Datenbussen miteinander verknüpft, so daß bis zu sechs Datentransfers gleichzeitig und simultan unter Mikrocode-Steuerung ausführbar sind. Die interne Datenbearbeitung ist somit relativ unabhängig und unterliegt keinen besonderen Einschränkungen bezüglich der Aufgabenstellung des Programmierers. **Bild 2** zeigt den internen Aufbau des Signalprozessors. Der hohe Grad an Parallelität der internen Pipelineregister (sechs Register) bietet eine beachtliche Flexibilität der Befehlsausführung. Die Ausführung von Befehlen ist simultan in fünf Ebenen möglich:

(1) Transfer mit externem Speicher
(2) Laden von Multiplizierer-Operanden
(3) Restaurieren von Produkten
(4) Übertragen von ALU-Operationen
(5) Datentransfers innerhalb des Registerfiles

Datentransfers im Registerfile sind durch ihre besondere Struktur gegeben. Jeder Register kann unabhängig von drei Quellen angewiesen werden. Zwei von ihnen (A1, B1) sind als Eingangs-Register (Eingabe) vorgesehen; die vier anderen dienen als Akkumulatoren. Die ALU des Am29501 bietet acht Funktionen:

R+S	R XOR S
R−S	R AND S
Pass R	$\overline{R}$
S−R	R OR S

Der Am29501 als Kern einer Signalprozessor-konfiguration besitzt zwei 8-bit-Ports zur Kommunikation mit den Multiplizierern Am29516 bzw. 17. MI0...MI7 dient zur Datenübernahme des höherwertigen Bytes vom Y-Port des AM29516/17. MIO0...MIO7 dient zur Übernahme des niederwertigen Bytes vom Multiplizierer, wie auch zur Übergabe von Daten an ihn. Ein dritter, bidirektionaler Port DIO0...DIO7 dient zum Anschluß des Datenspeichers der Abtastwerte. Die Eingänge I0...I28 des Am29501 dienen als Steuereingänge für den Anschluß des Mikroprogrammspeichers bzw. des Sequenzers Am29540. Der Mikroprogrammspeicher enthält die Information für alle Bussteuer- und Prozedurvorgänge im Prozessor. Die einzelnen Bitmuster steuern die ALU direkt, sie werden im Prozessor nicht weiter decodiert. Mit Hilfe des geeigneten Mikroprogramms ist der Am29501 vollständig autark gegenüber einem Hostrechner und führt, gemäß seinem Programm, einen numerischen Algurithmus aus.
Eine typische Applikation eines Signalverarbeitungsnetzwerks ist in **Bild 3** dargestellt. Die Multiplikationsfaktoren sind zweimal 16 bit. Da in der technischen Mathematik mit konjugiert komplexen Zahlen gerechnet wird, ist die Trennung zwischen Real- und Imaginärteil obligatorisch. Der Am29516 oder 17 liefert ein Produkt in zwei 16 bit breite Register mit MSP bzw. LSP-Inhalt in der außerordentlich kurzen Zeit von nur 40 ns. Nutzt man die Parallelstruktur bzw. die Kaskadierbar- ▷



△

Bild 4: Der Radix-2 Algorithmus ist aus dem FFT-Butterfly-Diagramm abgeleitet, der, jeweils mit Bitumkehrordnung am Eingang oder Ausgang, aus vier Abtastwert-Lese-, vier Partialprodukt-Schreib- und zwei Koeffizienten-Schreibvorgängen. Dazwischen erfolgen Multiplikation bzw. Skalierungen.

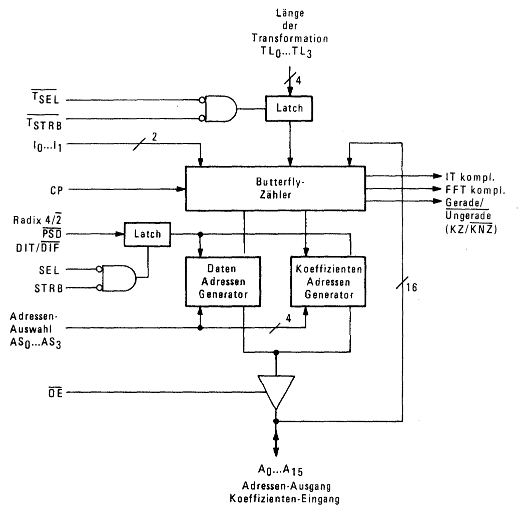


Bild 5: Blockschema des FFT-Adreß-Sequenzers Am29540 ▷

keit des gesamten 29500-Systems voll aus, werden für Real- und Imaginärteil jeweils zwei Slice-Prozessoren Am29501 eingesetzt. Als klassisches Beispiel für digitale Signalverarbeitungsanwendungen ist die schnelle Fourier-Transformation (FFT) zu nennen. Ihre Berechnungsformel für einen Amplitudenwert nach Betrag und Phase aus einer Abtastanzahl von 8 zeigt als Beispiel einen der Vektoren:

$$\overline{A3} = x_0W^0 + x_1W^3 - x_2W^2 - x_3W^5 - x_4W^0 - x_5W^3 + x_6W^2 + x_7W^5$$

$\overline{A3}$ ist das ausgeschriebene Polynom der Zeile 3 einer optimierten FFT-Matrix und ist als Beispiel hier beliebig herausgegriffen. Es ist eines der insgesamt 8 Vektorpolynome bei 8 Abtastwerten. Man sieht die Produktwerte, bestehend aus den Koeffizienten $x_0...x_7$ und den imganinären Faktoren (j*W*). Ein paralleles Zweiport-RAM ist erforderlich, um die Prozedurbandbreite zu garantieren. Man benötigt hierfür RAM-Bausteine z. B. des Typs Am93425A, um den außerordentlich hohen Datendurchsatz der 29500er Serie nicht zu beeinträchtigen.

Bezieht man sich auf die beschriebene Signalprozessor-Struktur, wird ein Radix-2 FFT-Butterfly-Algorithmus in 4 Befehlszyklen für einen Produktteil ausgeführt **(Bild 4).** Arbeitet man (wie häufig üblich) mit 1024 Signalabtastwerten (= 1024×1024 FFT-Butterfly-Matrix) berechnet die genannte Signalprozessor-Konfiguration mit dem Am29501 das konjugiert komplexe Amplituden-/Phasenspektrum in 3 ms. Das bedeutet, die schnellstmögliche Ausführung eines FFT-Algorithmus, die heute überhaupt auf Siliziumbasis denkbar ist.

Die ausgeführte Prozedur betrifft jedoch nicht nur den Datenpfad des Signalprozessors. Die FFT mit 1024 Punkten erfordert *in der optimierten Form* 5120 mal die 4-Zyklus-Prozedur, die insgesamt 15 360 Operanden- und Koeffizienten-Adressen benötigt. Diese Adressen werden mit Hilfe eines Adreß-Sequenzer-Bausteins produziert.

Der Adreß-Generator-Sequenzer

Der Organisator für die Verknüpfung der digitalen Abtastwerte mit den richtigen Prozedur-Koeffizienten in den jeweiligen Polynomen ist der Sequenzer-Baustein Am29540.

Bild 5 zeigt den internen Aufbau des Sequenzers. Die FFT-Prozedur kann für eine Zahl von 2...65536 Abtastwerten programmiert werden. Ebenso wird der Typ der Prozedur (Radix-2 oder Radix-4) an Pin 22 des 40poligen Bausteins festgelegt. Damit wird definiert, wie die Partialprodukte in der Formel (gemäß des Alorithmus) aufzuaddieren sind. Der Programmierer eines FFT-Alorithmus muß nun nicht mehr am absoluten Nullpunkt beginnen. Der Am29540 ist mit der Zuordnung der Partialprodukte gemäß des Butterfly-Diagramms bereits programmiert, und zwar mit den zwei am häufigsten verwendeten Verfahren Radix-2 und Radix-4. Der Programmierer hat also im wesentlichen nur noch das Prozedur-Mikroprogramm für die Addition bzw. Subtraktion der Partialprodukte in dem oben dargestellten Zeilenpolynom der FFT-Matrix zu schreiben. Dieses Mikroprogramm ist auf dem Entwicklungssystem SYS 29 zu entwerfen und dient dem Am29501.

Der Anfang der Abtastwertetabelle wird in Form der Offsetadresse dem Sequenzer übergeben. Dies wird üblicherweise von einem Hostrechner mit 8 oder 16 bit ausgeführt. Die Offsetadresse bietet die Verschiebbarkeit der Abtastwertetabelle, so daß nicht grundsätzlich bei Adresse 0 begonnen werden muß. (Adreßraum: 64 KB für Abtastwerte.)

Mehrebenen-Pipeline Register

Um die Verarbeitungseffizienz der ultraschnellen Am29501-Konfiguration optimal auszunutzen, wäre es wünschenswert, Speichertransfers mit laufenden Arithmetik-Operationen zu überlappen. In den genannten FFT-Alorithmen führt das praktisch dazu, den Speicher im *Read-Modify-Write*-Zyklus zu bedienen, d. h. ein Operand wird ausgelesen und nach der ALU-Verknüpfung im Schreibzyklus zurückgeladen. Um diese Verarbeitung zu sichern, wurden spezielle Mehrebenen Adreßregister entwickelt, die die Notwendigkeit einer zyklusabhängigen Adreßspeicherung gewährleisten. Die Bausteine im 24poligen Slimline-Gehäuse haben intern vier von ansteigenden Flanken getriggerte Register mit 8 bit, die im zweifach 2-Ebenen- oder einfach 4-Ebenen-Pfad auf Bedingung zusammenschaltbar sind. Die Raffinesse liegt darin, daß nach einem Adreßladevorgang die alte Leseadresse nicht verloren geht, sondern in die nächste Stufe als neue Schreibadresse für den Rückschreibvorgang immer noch zur Verfügung steht. Damit ist sichergestellt, daß nach Ausführung der Arithmetik die für diese Prozedur superschnelle Verarbeitung nicht durch eine primitive Adreßaufbereitung negativ beeinflußt wird. Die Adreßaufbereitung ist also in jeder Phase schneller als die Arithmetikphase.

Der Baustein Am29521 unterscheidet sich vom Am29520 in Einzelheiten des Ladevorgangs.

Zusammenfassung

Das Signalprozessor-Konzept der Serie Am29500 von *Advanced Micro Devices* stellt die absolut höchste Kategorie bipolarer VLSI-Bausteine dar, die bis heute geschaffen wurde. Die Anwendung eines kompromißlos schnellen Prozesses in Verbindung einer Schaltungstechnik in nicht mehr überbietbarer paralleler Struktur liefert bei äußerst rechenintensiven Prozeduren eine Verarbeitungsgeschwindigkeit, die von heute bestehenden Größtrechnern nicht zu überbieten ist. Im Prinzip ist auf der Größe zweier *Multibus*-Platinen ein Digital-Signalprozessor für FFT-Alogrithmen unterzubringen, der eine Rechengeschwindigkeit aufweist, die von einer *Cray1* (in herkömmlicher Technologie) geboten wird. *Advanced Micro Devices* wird diesem Konzept weitere Spezialbausteine dem hinzufügen.

ei 462

Literatur

[1] **New, Bernhard J.:** The Am29500 Signal Processing Family. *Advanced Micro Devices.*
[2] **Mick, J. R., New, B. J.:** Bit Slice Devices for Signal Processing. Proc. IEEE International Conference on Acoustics, Speech and Signal, April 1980.
[3] LSI Multipliers HJ Series. *TRW LSI Products.*

One-chip sequencer shapes up addressing for large FFTs

The addressing circuitry of a single IC accesses both data and coefficient memories for performing a broad class of fast Fourier transforms.

As one of the most useful algorithms in the digital signal-processing repertoire, the fast Fourier transform provides a quick, orderly, and convenient means of computing the frequency spectrum of a signal. When combined with other operations, the FFT is also useful in correlating or convolving two or more waveforms, techniques required to perform radar, sonar, and image processing.

One of the most difficult problems facing the FFT hardware designer is creating the circuitry to address the memories that hold the data variables and coefficient constants. The difficulty arises partially because of the memory space required and the resulting complexity of either accessing a large number of personalized address tables for each FFT or calling out a large data base in the proper sequence. Even when addressing is done in software, there is the problem of speed—the method is often too slow for real-time applications.

A new chip, however, contains all the addressing circuitry needed to access an FFT unit's data and coefficient memories so that a broad class of functions can be analyzed. The Am29540 programmable address sequencer is flexible enough to generate addresses for FFTs having as few as 2 or as many as 65,536 points. Twelve algorithms are supported in radix-2 and radix-4 systems, including operations on complex and real-valued input data (either in-place or non-in-place transforms); forward and inverse transforms; and decimation-in-time (DIT) and decimation-in-frequency (DIF) algorithms.

A web of nets

Included in the 16-bit sequencer are a butterfly counter (see "Generating Addresses Efficiently," p. 160), a data address generator, and a coefficient address generator (Fig. 1). The butterfly circuit actually has two counters, one for columns and one for rows. The column counter points to the current FFT stage, or column; the row counter, to the butterfly currently being performed within that stage. The counters are programmable and can be initialized to perform transforms of various lengths by prestoring the appropriate 4-bit transform-length code in an on-chip latch. The transform-length code is placed on input lines TL_0-TL_3 and latched with signals

David Quong and **Robert Perlman**
Advanced Micro Devices Inc.

Robert Perlman is a senior product planning engineer with the DSP/array processing group at Advanced Micro Devices in Sunnyvale, Calif. He obtained a BSEE from the Rensselaer Polytechnic Institute and an MSEE from the Johns Hopkins University, and has previously done design work in airborne digital signal processing at Westinghouse.

David Quong is a product planning engineer with the DSP/array processing group. He received a BSEE from California State University in Sacramento.

Transform Select (TSEL) and Transform Strobe ($\overline{\text{TSTRB}}$).

The butterfly counter executes one of four instructions: Reset, Reset/Load, Count, and Hold. These instructions are selected with control lines I_0 and I_1 and are executed on the rising edge of the Clock Input line (CP). An FFT is begun by initializing the butterfly counter with a Reset or Reset/Load instruction. The Count and Hold instructions are then used to advance the counter to the next butterfly operation or to hold it at the present butterfly position.

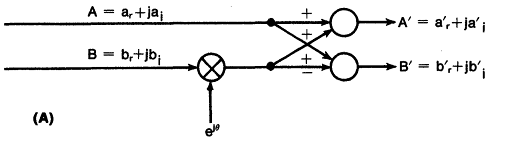
The counter section generates four flags to help control FFT sequencing. The Iteration Complete flag (IT COMP) indicates the last butterfly operation performed in a stage or column; the last butterfly operation in a particular FFT is signaled by the FFT Complete flag (FFT COMP). The Even/$\overline{\text{Odd}}$ flag changes

Generating addresses efficiently

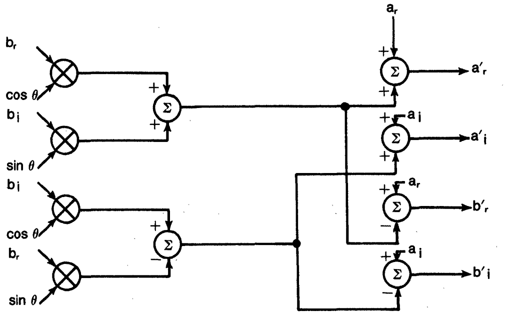
A quick look at the structure of a fast Fourier transform reveals why the data and coefficient circuitry is so complex. At the heart of the FFT algorithm is the butterfly operation, which takes its name from the schematic representation that shows how output data is generated from an input waveform.

In the butterfly operation on a radix-2 DIT FFT (Fig. A), two complex data points, A and B, and one complex coefficient are used to compute two new complex data points, A′ and B′. The coefficient is a complex exponential of the form $e^{-j\theta} = \cos\theta - j\sin\theta$. Each butterfly requires one complex multiplication, one complex addition, and one complex subtraction or four real multiplications, three real additions, and three real subtractions (Fig. B).

An FFT is performed by concatenating butterfly operations. The butterflies are arranged in columns, or stages; an N-point, radix -2 FFT comprises $\log_2 N$ stages, each containing N/2 butterflies, and so a total of (N/2) ($\log_2 N$) operations must be done.



(A)



(B)

The structure of a 16-point FFT contains 32 butterflies (Fig. C). Each circle represents a single radix-2 butterfly operation. The fractional number accompanying each butterfly of the last three stages is the coefficient needed to perform that butterfly: if the value of the fraction is k, the corresponding coefficient value is $e^{-j\pi k}$. Memory locations in which data is stored are represented as blocks; in the case of the 16-point FFT, 16 contiguous memory locations must be allocated to store the 16 complex data points. Each butterfly is performed by taking input points from the data memory, doing the necessary mathematical operations with the appropriate coefficients, and returning the results. The algorithm shown is in-place, meaning that the data points produced by each butterfly are stored in the same locations as the input data points.

The order in which data must be accessed is not straightforward. For the FFT shown, data must be accessed in order 0, 8, 1, 9, ... 7, 15, for the first stage. For the second and following stages, however, data addressing is somewhat more involved. The second stage, for example, is performed by accessing addresses 0, 4, 1, 5, 2, 6, 3, 7, for the first group of four butterflies; then 8, 12, 9, 13, 10, 14, 11, 15, for a second group. The butterflies in the third and fourth stages are also addressed in groups. Stage m has 2^{m-1} groups of $N/2^m$ butterflies with a group spacing of $N/2^{m-1}$

Coefficients must also be accessed. For the first stage of this FFT, only sin 0 and cos 0 need be acquired. Stage two, however, uses angles 0 and $\pi/2$; stage three uses $0, \pi/2, \pi/4$, and $3\pi/4$; and stage four needs angles $0, \pi/2, \pi/4, 3,\pi/4, \pi/8, 5\pi/8, 3\pi/8$, and $7\pi/8$. In general, the coefficient address sequence for the mth stage of this FFT is 0, $BR(1)\pi/N$, $BR(2)\pi/N$, ... $BR(2^{m-1}-1)\pi/N$, where BR(x) is a function that reverses the order of the bits of a binary word.

A new coefficient must be accessed for each group of butterflies. Other types of FFTs have various addressing sequence requirements, but this example is a good representative. FFT analyzers use several techniques to state after every stage and can be used to control memory operations for non-in-place transforms. The fourth flag, KNZ/$\overline{\text{KZ}}$, is of special use when performing transforms with real-valued inputs. The last two flags are multiplexed onto a single pin. When the sequencer produces a data address for a transform with a real-valued-input, KNZ/$\overline{\text{KZ}}$ appears on the pin; for any other type of data, Even/$\overline{\text{Odd}}$ appears.

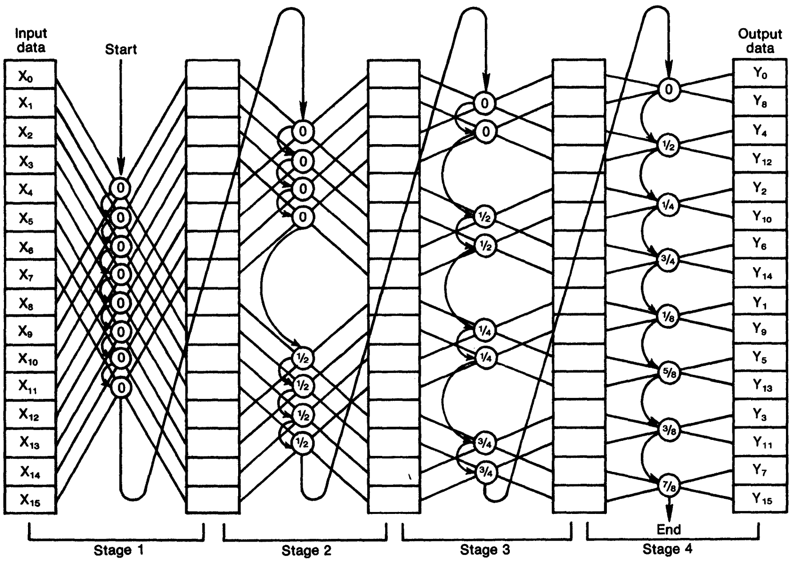
When performing a transform, the sequencer uses its address generators to create addresses combinatorially—that is, the data address generator produces an address for each input and output data point, and the coefficient address generator creates addresses for coefficients and weighting functions. Three control signals—$\overline{\text{PSD}}$, DIT/$\overline{\text{DIF}}$, and Radix4/$\overline{2}$—configure the address generators for various types of FFTs. These signals are stored in an

generate these data and coefficient addresses.

One of the most common solutions is to place the data and coefficient addresses in PROMs and then to sequentially address the PROMs with a counter. This approach has several serious drawbacks, however. First the number of data addresses becomes prohibitively large as the size of the FFT grows. A 4096-point, radix-2 FFT with complex inputs, for example, must address 24,576 butterflies, each requiring two data addresses and a coefficient address, for a total of 73,728 addresses. Although that number can be reduced by employing constant-geometry FFT algorithms that use the same data addresses for every stage, these algorithms have the disadvantage of being non-in-place, thereby requiring twice the data memory of an in-place transform. The second disadvantage of the PROM approach is that if a single system is to perform several different sizes or types of FFT, a different address table is needed for each FFT.

Another method uses as much SSI and MSI logic as is practical. This approach is easily implemented but usually results in a circuit that consumes considerable board space, is a headache to control, and takes a long time to debug. A circuit for the addressing function in a 4096-point FFT might require 10 to 20 chips.

A third approach is to compute the necessary addresses in software, a method that is often too slow for real-time applications.

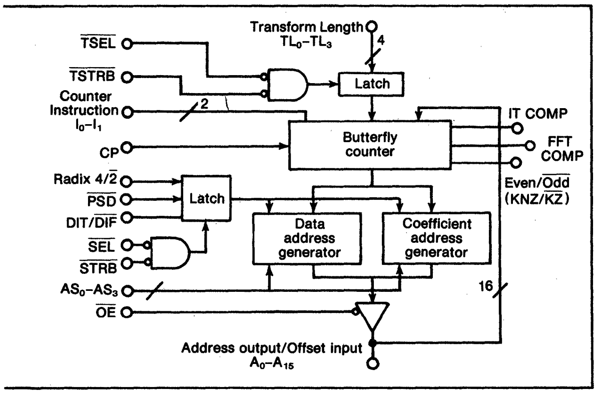


(C)

on-board latch controlled by the Select and Strobe lines ($\overline{SEL}$, $\overline{STRB}$). The user selects the desired input data, output data, or coefficient address with control lines Address Select 0 through 3 (AS_0-AS_3). The address chosen by lines AS_0-AS_3 is placed on Address lines A_0-A_{15}. Those lines can be forced to a high-impedance state by the Output Enable signal ($\overline{OE}$), thus allowing other address generation devices to be tied to the same address bus.

Conserving memory

When addressing data, the sequencer can generate as many as 2^{16} addresses; the actual number needed for a particular FFT depends on the size and type of the transform. An N-point, radix-2, in-place FFT with inputs that are complex quantities, for example, must address N complex-data points during each stage. If N is 16, only 16 memory locations need be addressed, leaving much of the available address space unused.



1. The Am29540 offers a one-chip solution to the problem of addressing data and coefficient memories for performing fast Fourier transforms. The butterfly counter can be programmed to address anywhere from 2 to more than 65,000 points.

The Am29540 offers two data-addressing options for the user who needs less than 64 kwords of space. The first sets the unused upper address bits to zero by initializing the butterfly counter with the Reset instruction. For a 16-point transform, then, the upper 12 address lines would contain 0s for any data address. The four remaining lines are available to call 2^4, or 16, values. The other option is to program the upper address lines to a user-selected value to address a given data block in a large memory. The upper data bits are programmed by bringing $\overline{OE}$ high, placing the desired bit pattern on address lines A_0-A_{15}, and then executing the butterfly counter's Reset/Load instruction. If, for example, the bit pattern $ABC0_{16}$ is used to initialize a 16-point, complex-input FFT, the sequencer will address a block of sixteen data locations beginning at address $ABC0_{16}$.

Non-in-place transforms present additional problems. Unlike in-place transforms, non-in-place algorithms cannot store the output data from a butterfly operation in the same locations previously occupied by the input data. That problem is overcome by generating both the input and output data addresses for such transforms.

Typically, non-in-place transforms are performed with two data memories, one the source of input data, the other the destination for output data. When a butterfly operation for a given stage is completed, the roles of these memories are reversed, with the output data memory of one stage providing the input data for the next. The Even/$\overline{Odd}$ signal is particularly useful in such cases; since it changes state after every stage, it can be used to control the direction of data flow between the two memories.

Getting the coefficients

To access coefficients, the Am29540 generates a 16-bit address corresponding to one of 2^{16} equally-spaced angles between 0 and 2π radians. For coefficient address A, the angle addressed is $2\pi A/2^{16}$ the angle $\pi/2$, for instance, would have the address 4000_{16}. The coefficient address is fed to look-up memory, usually PROM, containing sine and cosine values for the angles selected.

A given FFT will use some subset of the more than 65000 angles available. As a case in point, an N-point, radix-2 FFT with complex inputs must access only N/2 equally spaced angles in the range 0 to π radians; a 16-point FFT, then, needs only eight different angles. The sequencer automatically chooses the angles needed in the proper sequence, skipping over unused values.

The coefficient-addressing scheme employed carries a significant benefit for systems in which various sizes of FFTs are to be implemented. Because the chip automatically accesses only those sine and cosine values needed, a single sine/cosine table can be used to perform FFTs of various sizes. If, for example, the user creates a look-up table containing 2048 sine and cosine values between 0 and π radians, that table can be used to perform all radix-2 complex FFTs with 4096 or fewer points.

Most FFT algorithms currently in use are designed to process complex input data. The Am29540 supports 12 different types of this transform (see the table, below). The choices include:

•Radix-2 or radix-4 transforms. The butterfly structure of a radix-4 transform is more complicated than that of radix-2 but offers somewhat greater computational efficiency. Each radix-4 butterfly produces four output data points from four input data points and three coefficients, and consumes 12 real multiplications and 22 real additions. Radix-4 transforms are selected with the Radix4/$\overline{2}$ signal.

•Decimation-in-time or decimation-in-frequency transforms. These terms refer to two basic classes of FFTs; they reflect the manner in which each class is derived. DIT and DIF

Fast Fourier transforms supported by the Am29540

Input data type	Radix	Decimation type	In-place/ non-in-place	Input data ordering	Output data ordering	Direction of transform
Complex	2	DIF	In-place	Normal	Digit-reversed	Forward and inverse
	2	DIF	In-place	Digit-reversed	Normal	Forward and inverse
	2	DIF	Non-in-place	Normal	Normal	Forward and inverse
	2	DIT	In-place	Normal	Digit-reversed	Forward and inverse
	2	DIT	In-place	Digit-reversed	Normal	Forward and inverse
	2	DIT	Non-in-place	Normal	Normal	Forward inverse
	4	DIF	In-place	Normal	Digit-reversed	Forward and inverse
	4	DIF	In-place	Digit-reversed	Normal	Forward and inverse
	4	DIF	Non-in-place	Normal	Normal	Forward and inverse
	4	DIT	In-place	Normal	Digit-reversed	Forward and inverse
	4	DIT	In-place	Digit-reversed	Normal	Forward and inverse
	4	DIT	Non-in-place	Normal	Normal	Forward and inverse
Real-valued (RVI)	2	DIT	In-place	Normal	Unique	Forward
	2	DIF	In-place	Unique	Normal	Inverse

butterfly structures differ somewhat but require an identical number of arithmetic operations. The DIT/$\overline{\text{DIF}}$ signal determines the desired transform type.

•In-place or non-in-place transforms. Non-in-place transforms require twice the data memory of their in-place counterparts. It might seem, then, that in-place transforms would always be preferred. Unfortunately, the in-place approach has a drawback—the digits of the address of the input or output data must be called for in reversed order. This scheme requires a reordering operation. The choice between in-place and non-in-place algorithms is made by using the appropriate values of A_0-AS_3 to select the desired addresses. Should the user select an in-place transform, the choice of digit-reversed-address input or output can be made with the signal $\overline{\text{PSD}}$.

Useful inversions

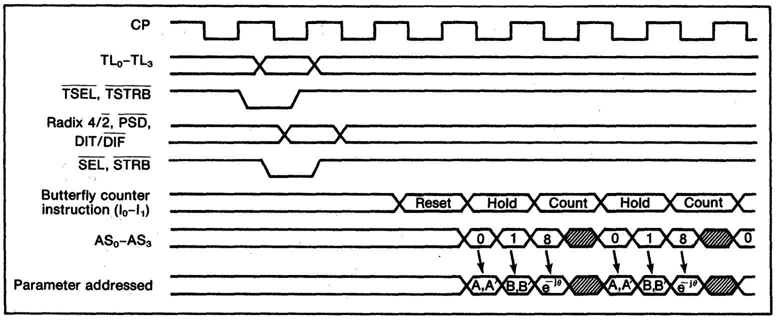
The sequencing chip also can be used to perform inverse transforms, a useful feature in applications requiring a route from the frequency to the time domain. Computing inverse transforms is straightforward—the address sequences needed are the same as those for the forward operations. With radix-2 transforms, the only difference between the inverse and forward transforms is the complex exponential: $e^{-j\theta}$ must be replaced with $e^{j\phi}$. Changing the sign of the complex exponential's argument is equivalent to replacing the coefficient sin θ with $-$sin θ, an operation that can be executed by slightly modifying the addition and subtraction operations performed in the butterfly. Radix-4 inverse transforms require somewhat similar minor accommodations to sign changes in the butterfly calculation.

Some applications demand FFT transforms with real-valued inputs. The sequencer generates data addresses for both forward and inverse real-valued-input (RVI) transforms of a type first described by Bergland.[1]

A weighty matter

FFT filter characteristics can often be significantly improved by premultiplying the input data with a series of weighting factors. This technique, also called windowing, or shading, can significantly lower filter sidelobes and thus simplify the analysis. The properties of a number of common weighting functions are well-documented.[2]

The sequencer supports two weighting approaches for radix-2 transforms. The first and


CP
TL_0-TL_3
$\overline{\text{TSEL}}$, $\overline{\text{TSTRB}}$
Radix 4/$\overline{2}$, $\overline{\text{PSD}}$, DIT/$\overline{\text{DIF}}$
$\overline{\text{SEL}}$, $\overline{\text{STRB}}$
Butterfly counter instruction (I_0-I_1)
Reset
Hold
Count
Hold
Count
AS_0-AS_3
0
1
8
0
1
8
0
Parameter addressed
A,A'
B,B'
$e^{-j\theta}$
A,A'
B,B'
$e^{-j\theta}$

2. The Sequencing operation for a 16-point FFT begins by loading the appropriate transform-length code and control signals into on-board latches. The butterfly counter is then reset. After initialization, the first butterfly's memory addresses are selected with lines AS_0-AS_3. The sequencer is then advanced to each succeeding butterfly using the count instruction.

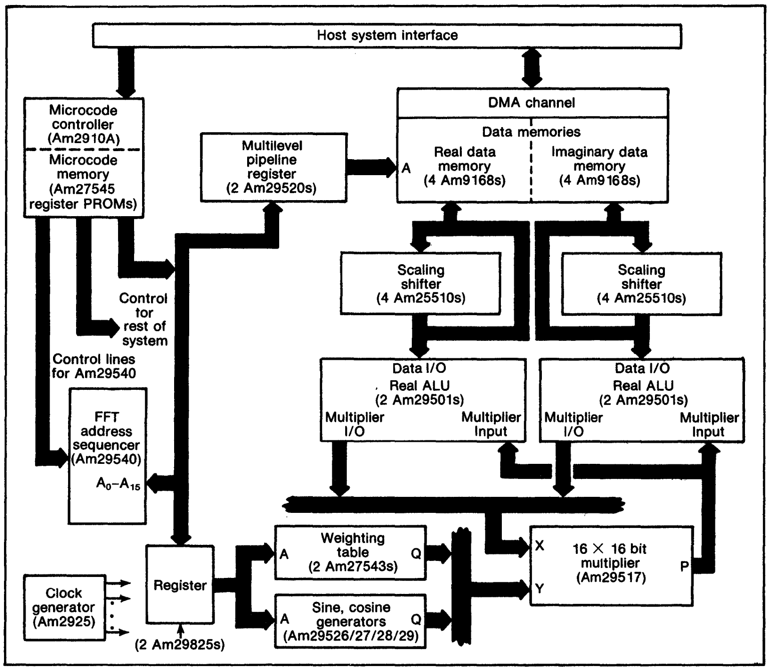
simplest approach is to perform a weighting prepass before the FFT begins.

The sequencer is programmed to perform the first stage of a radix-2 DIF transform. The resulting prepass data addresses access the input data, the coefficient addresses access weighting values stored in a look-up table. On completion of the prepass, the part is reprogrammed to address the type of radix-2 FFT desired.

The second approach takes advantage of the structure of a DIT FFT. For the first stage of the transform, only the coefficient values sin 0 and cos 0 are needed. Weighting can thus be incorporated in this first stage, using the stage's multiplier. By configuring the part to perform a radix-2, DIF FFT for stage 1, and then changing the FFT type from DIF to DIT for all remaining stages, the necessary data, weighting, and coefficient addresses can be generated.

A look at an FFT

Virtually all useful weighting functions are symmetrical. If Y(n) is a symmetrical N-point weighting function, point Y(x) is equal to Y(N−x). This symmetry implies that the user need not store all N points of the weighting function: (N/2)+1 points are sufficient. The sequencer addresses such half tables by generating both x and N−x. The often-used von Hann weighting function is one such example,


Host system interface
Microcode controller (Am2910A)
Microcode memory (Am27545 register PROMs)
Multilevel pipeline register (2 Am29520s)
DMA channel
Data memories
A
Real data memory (4 Am9168s)
Imaginary data memory (4 Am9168s)
Scaling shifter (4 Am25510s)
Scaling shifter (4 Am25510s)
Control for rest of system
Control lines for Am29540
Data I/O
Real ALU (2 Am29501s)
Multiplier I/O
Multiplier Input
Data I/O
Real ALU (2 Am29501s)
Multiplier I/O
Multiplier Input
FFT address sequencer (Am29540)
A_0–A_{15}
Weighting table (2 Am27543s)
A
Q
X
16 × 16 bit multiplier (Am29517)
P
Y
Clock generator (Am2925)
Register
(2 Am29825s)
Sine, cosine generators (Am29526/27/28/29)
A
Q

3. In a typical system, the Am29540 is used to access data, weighting, and sine and cosine (coefficient) values in a microcode-controlled FFT processor. This system can support a 4-kpoint radix-2 transform.

One-chip FFT sequencer

easily derived from the table of cosines required by the FFT algorithm itself. Thus, the need for a separate weighting-function memory is altogether eliminated.

The sequencer's operation can be best understood by considering its performance of a typical FFT. Suppose, then, that an in-place, radix-2, 16-point DIT FFT is to be implemented (see "Generating Addresses Efficiently," Fig. C, p. 160). To initialize the device, the appropriate transform-length code and control bits are loaded into the on-chip latches. For this example, the transform-length code has the value 0011_2; the control bits must assume the values $\overline{PSD} = 1$, Radix4/$\overline{2} = 0$, and DIT/$\overline{DIF} = 1$. After this data has been entered, the butterfly counter is initialized with a Reset or Reset/Load instruction.

Once initialized, the part generates data and coefficient addresses for the FFT's first butterfly. For this algorithm, the input and output data addresses are set at 0 and 1, respectively, with lines AS_0-AS_3; the coefficient address is similarly set to 8. After all the addresses have been read, the device is advanced to the next butterfly by executing a count instruction (Fig. 2).

Defining the system

The Am29540's working environment is a microcode-driven FFT processor. That system can be divided into several basic blocks (Fig. 3): the address sequencer, arithmetic processor, high-speed data memory and coefficient memory, system controller, and the host interface.

The address computer generates the read and write addresses to access data, as well as coefficient and weighting addresses.

The arithmetic processor, consisting of a multiplier (here, the Am29517) and two ALUs (one for real and the other for imaginary data), efficiently calculates complex data from the data memory. Using coefficient and weighting generators, it processes the information and returns it to the data memory. The data width is 16 bits; therefore each ALU requires two 8-bit-slice multiport pipelined processors (here, Am29501s). A scaling shifter is provided in each data path from the memory to the ALUs.

The high-speed data memory stores input and output data from an FFT operation. It is also divided into two banks, one for real, one for imaginary data. The sequencer can be loaded with a data address offset, allowing data memory to be addressed at starting locations other than zero, and permits the addressing of selected blocks of data. A set of coefficient generators (Am29526/27/28/29) provide the coefficients needed when performing an FFT and produce up to 2048 words of sine and cosine data. This is sufficient to support up to a 4096-point, radix-2 transform. A PROM contains the weighting values for the FFT input data.

The system controller, as overseer, accepts instructions through the host computer interface, determines which function must be performed, issues the proper instructions to other components, and informs the host when the operation is done. It employs a microsequencer (the Am2910A) and microcode memory.

The host interface consists of logic to handle the host system protocols and a DMA controller for high-speed data transfer. During block data transfer the DMA circuitry has direct access to the data memory.

The address sequencer generates both read and write addresses for the data memory. When, as is usual, the operations for a sequence of butterflies are overlapped, those addresses must be temporarily stored in an agile shift-register pipeline. This structure must unravel the intertwined sequence of addresses for the several butterflies that are in progress at any given time. Here, a multilevel pipeline register consisting of two Am29520s is used. It can serve as dual two-level or a single four-level pipeline register, and each of the registers is available to the output at any time. □

References

1. G. D. Bergland, "A Fast Fourier Transform Algorithm for Real-Valued Series," *Communications of the ACM,* Vol. 11, Number 10, October 1968, pp. 703-710.

2. F. J. Harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proceedings of the IEEE,* Vol. 66, Number 1, January 1978, pp. 51-83.

CUSTOMIZABLE ANALOG CIRCUITS USING DIGITAL SIGNAL PROCESSING

by Russell J. Apfel
Advanced Micro Devices
9Ø1 Thompson Place
Sunnyvale, CA 94Ø88

INTRODUCTION

Customized and semi-customized analog circuits are used in many applications in place of standard products in order to achieve a specific analog function with reduced cost, board space, and power dissipation. There are a number of different classes of applications and different levels of customization possible in implementing analog functions. Traditionally, customization has been achieved through custom or semi-custom designs using analog gate array techniques. The world of digital electronics also requires customization but much of this customization has been replaced by the use of programmable microprocessors. The analog equivalent of the microprocessor is the digital signal processing unit with digital programmability being used to replace analog customization.

DIGITAL SIGNAL PROCESSING PRODUCTS

Customizing products can be done at a number of levels depending on the system requirements. Some products can be designed for a generic type of function and then customized to allow different types of the generic product to be implemented. An example of this would be an FSK modem where the same functional blocks are required to implement most low-speed FSK modems and the customization comes about in the modifying design of these blocks to implement a specific modem. Digital Signal Processing can replace these types of customized analog designs by building a generic FSK modem and storing the characteristic of different specific modems within ROM in the Digital Signal Processing modem. The user can then select the section of ROM for the desired modem and realize achieve different types of modems with the same signal processor. This class of programmability/customization uses selection of different on-board ROMs to achieve desired customized results.

A second level of customization can be described by an analog function where the signal processing functions are well known but the characteristics must be customized for each individual application. A digital signal processing unit, with on-board RAM can be used where the RAM stores the characteristics that define the applications specific requirements. This RAM can be programmed for each specific application eliminating the need for special customized circuits.

The third class of customizable products is the general application where analog blocks are required but the specific connection of the blocks, number of blocks, and types of blocks is unknown. In this case, a general purpose signal processor - the analog equivalent of a microcomputer can be used. The type of analog blocks and their characteristics can be fully programmed. This general purpose approach can replace most complex analog sub-systems.

Digital signal processing can bring many new advantages to the system designers. Complex analog design can be replaced with DSP units that are programmed simply and easily using development systems and assemblers and the detailed analog design can be eliminated. Digital signal processors are finally becoming powerful enough and cost effective for use in replacing analog subsystems.

DIGITAL SIGNAL PROCESSORS

A Digital Signal Processor (DSP), as described within this paper, has four functional blocks as shown in Figure 1. The DSP unit consists of an A/D converter which is used to convert the analog signals into digital form, a digital to analog converter which is used to take the resulting digitized representation of an analog output and convert it back to analog, a digital system interface which allows the DSP

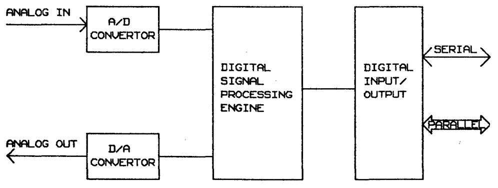


FIGURE1: DIGITAL SIGNAL PROCESSOR

unit to communicate with a microcomputer or other digital base system, and the digital signal processing engine. The digital signal processing engine shown in Figure 2

High Speed Multiply Filter

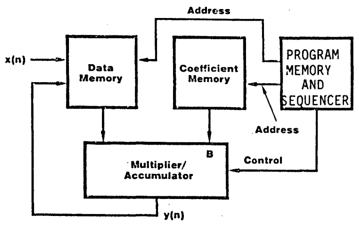


Figure 2: DSP Engine

consists of a arithmetic unit (a multiplier function and an accumulator), a set of control registers, a data memory for storing the incoming data, a coefficient memory which is used to store the signal processing characteristics and a sequencer. The sequencer includes a program memory (typically a ROM) as well as control logic for implementing the sequence. The digital signal processing system is not a simple system. For many years DSP systems were considered only in very expensive applications because they required a great deal of silicon to build them and they dissipated a great deal of power. With the advent of advanced MOS techniques, first in NMOS and today in CMOS, it is possible to build powerful digital signal processing units that can do useful functions at economic prices and with low power dissipation. The first cost effective signal processors are being used in application specific functions such as digital signal processing modems and digital signal processing subscriber line circuits. These are high volume functions which can be done with preprogrammed signal processors that are optimized for the specific function and are very cost effective. The more general purpose signal processors are just now starting to appear on the market with enough power and enough capability to do useful analog functions.

The remaining portions of this paper will describe some specific digital signal processing products, how they achieve customization for the user, and some comparison of performance. The areas to be discussed will be DSP modems, DSP Subscriber Line Circuits, and general-purpose DSP units.

DSP MODEMS

A modem (MOdulator/DEModulator) is a device which takes a signal and modulates it onto a carrier which can be used over a communication network and at the other end of the system takes the transmitted carrier and demodulates the signal to restore the original information. The most common type of modem is a voice band modem which takes digital information, and modulates it onto an analog signal carrier which is appropriate for transmission on a Public Switched Telephone Network. The most popular low cost modems use Frequency Shift Key (FSK) techniques for transmission of digital information on the public telephone network. This technique converts the incoming digital data into one of two discrete frequencies, a mark frequency which represents a zero or a one input and a space frequency which represents a zero input. Modulation between the two frequencies constitutes transitions between zero and one. The most popular FSK modems are 3ØØ-bit per second full-duplex modems using the Bell 1Ø3 standard in the United States, and the CCITT V.21 standards in Europe, and 12ØØ BPS half-duplex modems using the Bell 2Ø2 standard in the United States and the CCITT V.23 modem in Europe. Each of these popular types of modems use different frequency bands and have different signal processing requirements, but the generalized functions of these FSK modems are identical. Traditional customized analog designs require completely separate filter designs for each modem type and therefore separate chips are required to implement the different modems. The different modems also have different requirements for interfacing into the digital system. Specifically in Europe, the digital interface is the V.24 interface with special timing requirements. In the United States similar requirements are imposed by the RS232 interface. The use of digital signal processing allows significant simplification in the design of the modem.

The Am7910/11 WORLD-CHIP TM FSK modems from Advanced Micro Devices are examples of digital signal processing replacing customized analog design. The Am7910/11 uses a digital signal processor similar to Figure 1. The 7910/11 architecture is shown in Figure 3. The 7910/11 consists of a modulator or transmitter for a generalized FSK modem, a receiver or demodulator for a generalized FSK modem, a control and mode selection block which allows the user to select which type of modem he is implementing, and an interface handshake logic block which provides the interface into the digital system. The transmitter for an FSK modem is shown in Figure 4. The modem takes the incoming digital data, generates a sine wave using a digital sine wave synthesizer which selects frequencies by choosing different values from a ROM. The synthesized sine wave frequency is changed as a function of the incoming data. The synthesized signal must then go through a bandpass filter because the modulation process creates a wide spectrum of signal energy and energy outside of the band of interest must be filtered before transmitted on the telephone lines so as not to interfere with other signals. The characteristic of this digital bandpass filter is different for each modem type and a generalized filter is used with the filter coefficients selected from a ROM. In analog design, placing different filter banks would require excessive circuitry and would not be feasible. In the digital technology, the only additional circuitry required is additional ROM, and since ROM is relatively inexpensive, it is feasible to implement multiple modems on a single chip. The output of the bandpass filter is increased in frequency to a high sampling rate to eliminate the need for complex analog reconstruction filters and then fed into a digital analog converter and converted to an analog signal to be sent out on the telephone line.

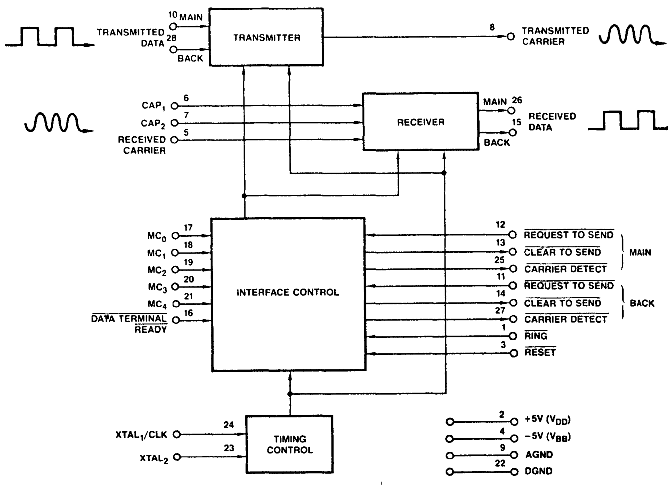


Figure 3: Am7910 Block Diagram

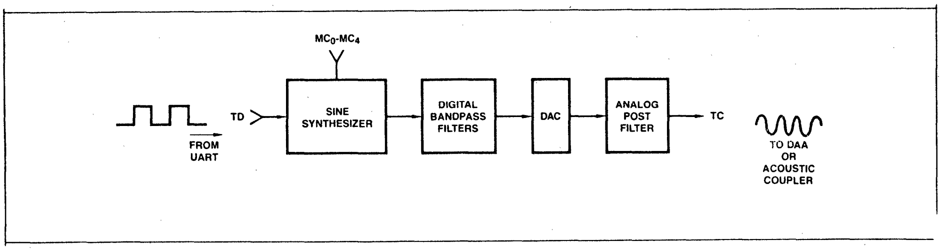


Figure 4: Am7910/11 Transmitter Block Diagram

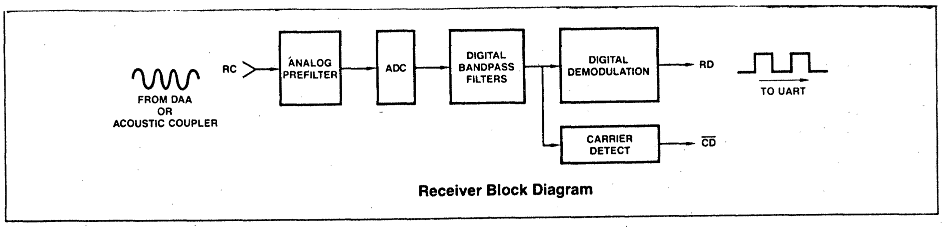


Figure 5: Am7910/11 Receiver Block Diagram

The receiver of the FSK modem is shown in Figure 5. The incoming analog signal goes through a simple low-pass filter and is converted to digital using a high sampling rate A/D converter. The output of the A/D converter is low-pass filtered and then fed into a digital bandpass filter to remove unwanted signals. The signal, after bandpass filtering, goes into several blocks. A carrier detect circuit is used to measure the energy of the incoming signal to determine whether there is a signal of large enough amplitude to denote a carrier on the line. This function usually requires full wave rectification, low pass filtering, and comparison with fixed threshold. The signal also goes to automatic gain control block which scales low level signals up to a constant level and then goes through a demodulation process. The FSK demodulation process in the Am7910/11 is called product or FM demodulation where the carrier is delayed by a fixed time delay (as a function of the modem type) and then multiplied by the undelayed incoming signal. This multiplication process gives a baseband (low frequency) term and a double frequency term. The double frequency term is removed by a filter and the remaining baseband signal goes into a decision device to detect the presence of a one or a zero. The digital bandpass filters, the delay in the demodulation process and the low-pass filters in the demodulation process are all customized for each modem application. These customized filter sections are stored in ROM and selected for the appropriate modem type in the Am7910/11.

The interface control logic on the Am7910 is a microprocessor like state machine which allows user selection of the various different timing signals to be generated for the modem system interface. The digital signal processing modem is a system, actually two systems, a modulator where the input is digital and the output is analog, and a demodulator where the input is analog and the output is digital. The required handshaking signal to meet various different international standards are digitally programmable to generate the various time delays as opposed to a customized or analog time delay scheme. The Am7910/11 modems have been in production for several years and are among the most successful modem devices on the market. They are the only devices that can do multiple speed modems within a single chip.

PROGRAMMABLE LINE CIRCUITS

Digital signal processing has been successfully used to implement a Subscriber Line Audio Processing Circuit (SLAC) which is used in telephone line interface circuit applications. The Am7901A SLAC includes a digital replacement for the analog codec filter plus six user programmable filters which allow customization of the telephone system under software control as opposed to using analog components.

Most telephone systems have different requirements for terminating the telephone line. The United States, for instance, requires an impedance of 900 ohms in series with 2.16 uf, whereas countries in Europe have different complex impedance. The German requirement is 100 ohms in series with 901 ohms and a parallel 86 nf capacitor, while in the United Kingdom there are different impedances required with short and long lines. These different impedances are usually implemented with external analog components that require reasonably good precision and have to handle high voltage. The digital filter in the SLAC provides a feedback which allows a simple fixed resistive termination to be modified under software control. Therefore, the single resistor can be used to simulate the US impedance, the German impedance, or the British impedance, all under software control. A single card can be designed that can be customized under software control of the system to be used in all the different applications without requiring a new design.

A block diagram of the SLAC is shown in Figure 6. The six user programmable filters allow the gain and the frequency response to be adjusted in both transmit and receive paths and two feedback filters which provide programming of the 2-wire input impedance system and the transhybrid balancing. Most telephone systems have the same basic line interface functions, but because of different administrations and different requirements, and different applications, different analog

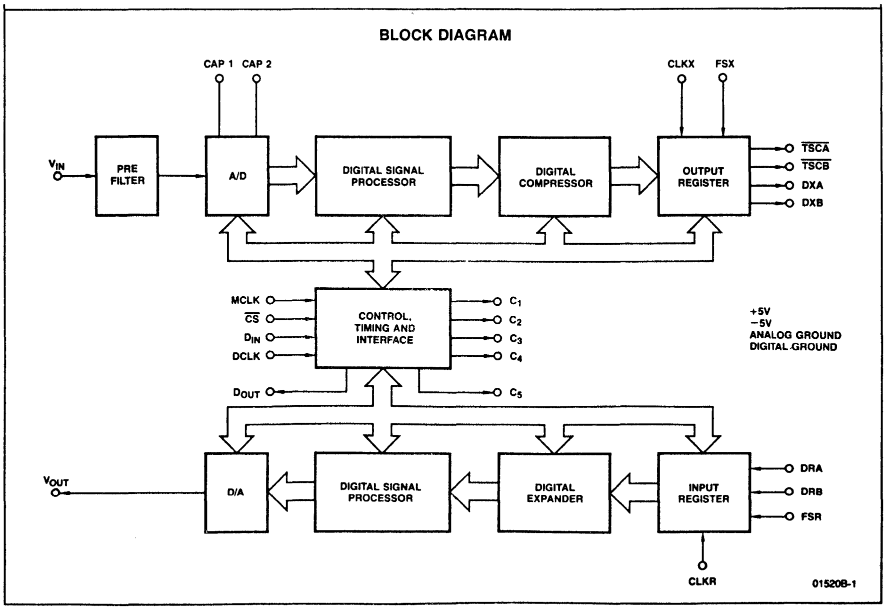


Figure 6: Am7901A (SLAC) Block Diagram

conditions must be set up. In traditional analog systems, the gain is set with discrete resistive components or an analog gain adjustable amplifier is required with custom design usually to allow computer control gain adjustments. Most analog systems do not have the ability to adjust, or compensate for frequency response variations. The SLAC with its programmable digital filters has this capability. The two feedback filters provide significant performance improvement and component reduction over analog systems.

Telephone line circuits also require a transhybrid balancing function. This is a function which cancels the echo generated on the 2-wire interface and feeding back around the system. Normally this is done by using an analog impedance that approximates the impedance at the telephone line to generate a balancing or canceling circuit. Because the telephone lines vary so much from line to line and area to area it is very difficult to achieve good balancing and typically echoes in the range of 10 to 30% of the signal are fed back around the echo path. Many telephone systems require that several networks be placed on a board and selected upon installation and testing. The selection is either done manually by throwing switches or automatically by testing and switching in and out analog networks. With a signal processing device like the SLAC, the balancing network is stored as a set of coefficients in a digital filter which emulates a response to the telephone line. Because a number of filters can be stored in a microprocessor and selected under control of a microprocessor, a much wider range of impedance is possible. The filter has the capability of matching a multi-pole system and replacing a number of resistors and capacitors with much higher performance. Future systems will use fully adaptive digital algorithms to obtain very significant improvements in balance where the echo path will be reduced to less than 1% echo. This improvement in performance again comes with reduction in external components and cost savings. The additional benefit of doing these functions digitally is that there is no human interaction. It is all done under computer control and of the software. Because there are no switches and no external components reliability of the system is significantly improved.

The digital signal processing capabilities of the SLAC make it a truly third generation telephone line circuit. The SLAC has been designed in to a number of the latest most modern telephone systems offering significant performance and improvement over that which is achievable with standard customized analog designs.

GENERAL PURPOSE DIGITAL SIGNAL PROCESSORS

The ultimate customized digital signal processor is shown in Figure 7. This is a general purpose signal processor with analog I/O where the sample rate is selected under user control, the user has a data memory available, a coefficient memory available, and a program memory. The analog digital system can be configured and reconfigured to do a number of different applications. This system can be programmed with a single program to do a specific application or under

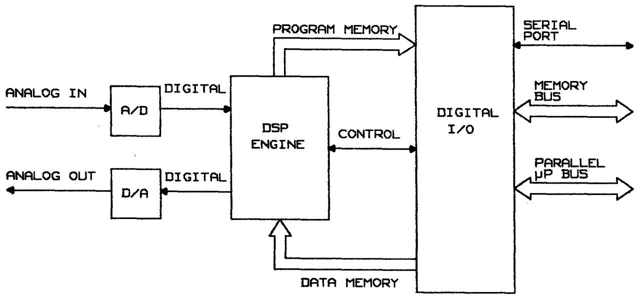


FIGURE 7: GENERAL PURPOSE DSP DEVICE

system control could have multiple programs and be reconfigured and reused to do a variety of different analog features. Two examples of very different application areas will be used to illustrate the power of this system.

The first application is in a process control system where there are a number of transducers (shown in Figure 8) and are connected to the digital signal processor. An external multiplexer is used to step through the different analog inputs. The analog input is converted to digital by the signal processor, the resulting signal is filtered to remove high frequency noise as well as power line noise. The linearization correction curve for the transducer (which is usually non-linear) is placed within the digital signal processor to correct for non-linearities within the system. The signal processor uses the input signal, for instance the input from a transducer that measures the flow rate of liquids flowing through a control valve, calculates out the flow rate, compares it to the programmed or the limit on the flow rate and generates an analog signal to control the valve or stepper motor or some actuator to increase or decrease the flow rate. This feedback control system then is completely under the control of the digital signal processor. With an analog system, a complete custom design would be. required for each actuator or each different type of valve, transducer and sensing system in the area. The high speed performance of digital signal processor allows it to interface to a number of different input sensors and output actuators all under control of the signal processor. This is a significant savings in cost of implementation and allows very simple updates and changes. If new transducers are used, new calibration and linearization curves can easily be put into the signal processor. In fact, the signal processor can go through a calibration routine to calibrate and calculate the linearization curve for the transducer, store it in memory and use this for all future calculations. The closed loop feedback system that is implemented has significant performance advantages over an analog system.

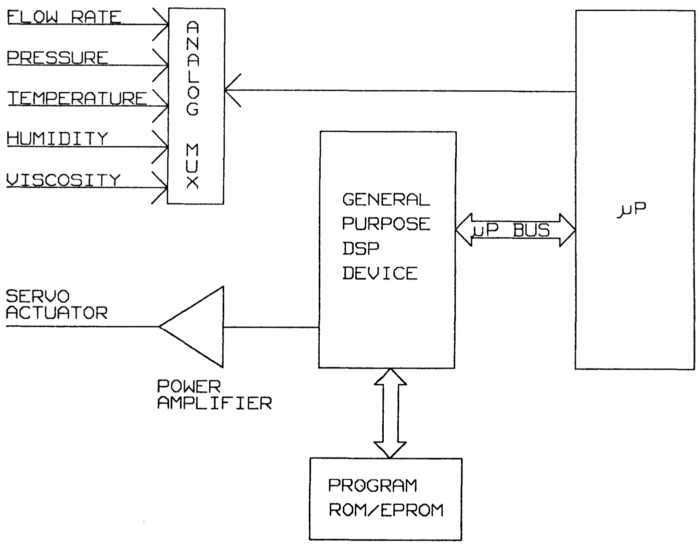


FIGURE 8: CONTROL SYSTEM WITH DSP DEVICE

Figure 9 shows a speech processing system using a general purpose signal processor. This speech processing system can be in a number of different applications. Speech can be compressed if an incoming signal is digitized and the result is applied to a digital processing algorithm to convert it to a low bit rate speech. The resulting output in digital form can be stored within local memory, can be transmitted over communications link for low bit rate communication and/or can be transmitted and then stored. The compressed data can be fed back in digitally to the signal processor which would restore the original analog signal from the compressed data and provide it to an output. This speech compression system can also be used to take the low bit rate speech or any processed speech and encrypt or scramble it in such a form as to implement a secure communications system. The analog equivalent of such a system would be a very complex custom design. One of the additional advantages of the digital signal processing approach is that as new algorithms develop and new systems develop they could easily be implementable with just software changes. Since these systems are very difficult to design and rarely work the first time, having to change ROM code would significantly reduce the development time and allow the products to be developed much more efficiently.

Speech signal processing systems can also be used for speech recognition. Speech recognition is an area just in its infancy and new advances are being made every day. The incoming analog signal would be digitized by the A/D converter and fed into the signal processor. The signal processor performs various times of filtering and comparison functions such as time domain warping and stretching of the signal. Comparisons with stored words or phrases can be used to do speaker dependent recognition. The simple signal processor can be augmented with additional memory and RAM and can be cascaded with other signal processors if required to provide the level of power necessary to implement the speech recognition algorithm. Speech recognition is an area which does not seem to be possible with standard analog techniques and the future for this area is dependent on digital signal processing.

CONCLUSION

Digital signal processing is just emerging as a cost effective alternative to analog design. The use of digital programmable signal processors to replace analog components is analogous to the effects that microprocessor systems have had on replacing customized digital designs. The level of complexity where it becomes cost effective to implement

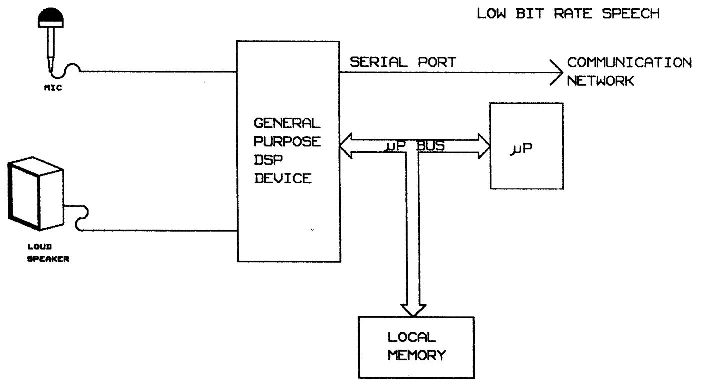


FIGURE 9: GENERAL PURPOSE DSP DEVICE IN SPEECH COMPRESSION/EXPANSION APPLICATION

programmable digital signal processing systems is constantly being reduced as more advanced technology and more advanced algorithms are developed for use on digital signal processors. Customized analog designs will never disappear but will continually be pushed into the realm of either very simple low cost designs or very high frequency high performance designs.

500-kHz single-board FFT system incorporates DSP-optimized chips

VLSI devices optimized for digital signal processing can realize architectures that, compared with traditional designs, save space, power and money. Such chips serve as the basis for a single-board system that uses fewer than 40 standard components.

Robert Cohen *and* **Robert Perlman**,
Advanced Micro Devices Inc

By employing VLSI devices to implement the fast Fourier transform, you can build a single-board digital-signal-processing system that supports sampling rates to 500 kHz and requires fewer than 40 packages (including processor, sequencer and local memory).

In such systems, the FFT makes possible many applications that would otherwise be unrealizable because of computational complexity. FFT techniques require a great number of calculations, and general-purpose computers incorporating the FFT aren't fast enough for such real-time high-bandwidth signal-processing systems as radar, video processing and telecommunications. Until the introduction of VLSI devices that are optimized for DSP tasks, only expensive array processors and special-purpose systems constructed with hundreds of SSI and MSI components could serve such applications.

Optimize butterfly execution

Effectively applying these VLSI circuits requires a familiarity with the FFT's computational requirements (see **box,** "FFTs reduce DFT computations"). Then, you can implement an appropriate algorithm in hardware. Because the FFT's basic operation is the butterfly, you can start by designing a butterfly processor.

Fig 1 lists the steps required to process a butterfly. The list helps you to determine the minimal resources required: an ALU, a multiplier and enough memory to hold the real and imaginary components of N samples. By adding resources, you can increase parallelism and boost throughput. For example, separate memories for the real and imaginary components of the sample data allow you to read A or B (or write A′ or B′) in one cycle. Extending this concept, you can divide the data pathway into a real-variable processor and an imaginary-variable processor **(Fig 2).**

The multiplier (or set of multipliers) acts as a shared

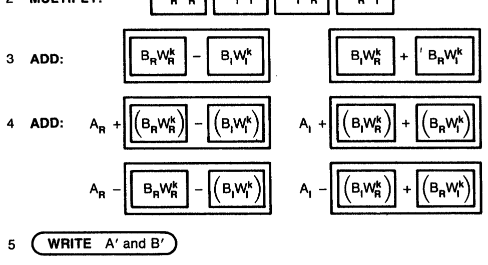


Fig 1—Five sequential steps *implement the butterfly, which is the primitive DFT operation. This list shows that the absolute minimum resources required to implement a butterfly are an ALU, a multiplier and memory.*

resource for both processors because it operates on both types of data. Each processor consists of an ALU and registers that hold intermediate results. Although you could add more ALUs, they prove superfluous for the FFT algorithm used here. (Additional ALUs are useful in radix-4 algorithms; see **reference.**)

To achieve the best performance, you minimize the number of cycles needed to execute the butterfly. Parallel computations allow the processor to accomplish more in each cycle to effect the desired reduction.

With an architecture like the one suggested—two memories, two processors and several multipliers—what is the smallest number of required cycles? To find out, examine **Fig 1** and start by using two cycles to read A and B from memory. (You can store W^k in a PROM and read it concurrently with A and B.) Assuming the processor has four multipliers, step 2 executes in one cycle. Step 3 also executes in one cycle if it determines the left-column difference in the real ALU and the right-column sum in the imaginary ALU. Step 4 requires two cycles, the left two operations being performed in the real-variable ALU and the right two in the imaginary-variable ALU. A′ and B′ are then written to memory in two cycles. This process executes a complete butterfly in eight cycles.

You now estimate how fast this processor can operate. Assuming that N=1024, the processor must perform $(N(\log_2)N)/2$ butterflies (a total of 5120). At eight cycles per butterfly, the processor needs 40,960 cycles. Next, assume a 100-nsec cycle time. (Cycle time depends on the slowest pathway through the system, which is typically via the multiplier; 16×16-bit combinatorial multipliers with sub-100-nsec propagation delays are common.) Under these conditions, a 1k-sample transform requires 4 msec, corresponding to a 0.25-MHz sampling rate, which is quite respectable for many applications. Further scrutiny will reveal ways to reduce hardware and increase throughput.

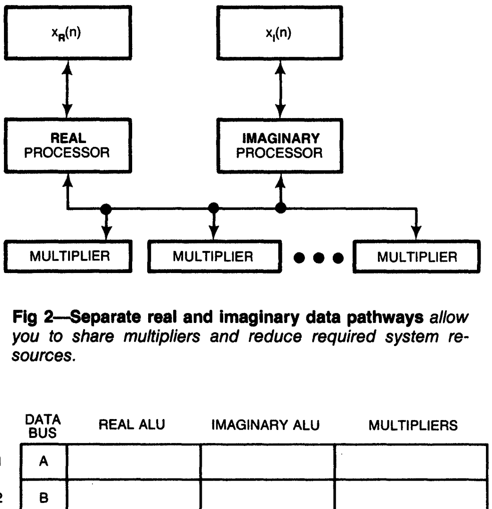


Fig 2—Separate real and imaginary data pathways *allow you to share multipliers and reduce required system resources.*

Less hardware does the job faster

Fig 3 shows a resource-utilization table for an 8-step butterfly. Note that all resources are idle most of the time: The data bus is active only 50% of the time, the ALUs 38%, and the multipliers 13%. You can take advantage of this idle time by executing butterflies concurrently, a technique known as pipelining.

For example, after reading A and B for the first butterfly, the data bus can read A and B for the second butterfly during cycles 3 and 4 while the multipliers and ALUs are busy. The multiplier could then begin working on the second butterfly immediately after computing results for the first. Using this technique, the processor still requires eight cycles to complete a

	DATA BUS	REAL ALU	IMAGINARY ALU	MULTIPLIERS
1	A			
2	B			
3				$B_RW_R^k$ $B_IW_I^k$ $B_RW_I^k$ $B_IW_R^k$
4		$B_RW_R^k - B_IW_I^k$	$B_IW_R^k + B_RW_I^k$	
5		$A_R + B_RW_R^k - B_IW_I^k$	$A_I + B_IW_R^k + B_RW_I^k$	
6		$A_I - B_IW_R^k - B_RW_I^k$	$A_R - B_RW_R^k + B_IW_I^k$	
7	A′			
8	B′			

Fig 3—An 8-step butterfly, *which implements the algorithm prior to optimization, uses its constraining resource—the data bus—only 50% of the time.*

	DATA BUS	REAL ALU	IMAGINARY ALU	MULTIPLIERS
1	A			
2	B			
3				$B_RW_R^k$ $B_IW_I^k$ $B_RW_I^k$ $B_IW_R^k$
4		$B_RW_R^k - B_IW_I^k$	$B_IW_R^k + B_RW_I^k$	
5	A	$A_R + B_RW_R^k - B_IW_I^k$	$A_I + B_IW_R^k + B_RW_I^k$	
6	B	$A_I - B_IW_R^k - B_RW_I^k$	$A_R - B_IW_R^k + B_RW_I^k$	
7	A′			$B_RW_I^k$ $B_IW_I^k$ $B_RW_I^k B_IW_R^k$
8	B′			

LOOP ON THESE STEPS

Fig 4—By starting a second butterfly concurrently, *you can create a 5-cycle loop and improve throughput 38% compared with **Fig 3's** operation.*

particular butterfly, but it reduces the average number of cycles per butterfly because it works on more than one butterfly at a time.

The most heavily used resource determines the minimum average number of cycles per butterfly that you can achieve. By using the four idle bus cycles, you can reduce the average number of cycles per butterfly to four and double system throughput.

You can see this doubling of throughput clearly in **Fig 4**'s resource-utilization table. A second concurrent butterfly starts on cycle 5; A and B are then read, and the new products are calculated in cycle 7. (You can also start the second butterfly on cycle 3 or 4.) After completing cycle 8, the processor jumps to cycle 4 instead of cycle 1, because it has already read A and B and computed the new products for the second butterfly. This technique creates a 5-cycle loop instead of an 8-cycle loop, improving throughput by 38%.

	DATA BUS	REAL ALU	IMAGINARY ALU	MULTIPLIERS	
1	A				
2	B				
3				$B_RW_R^k$ $B_IW_I^k$ $B_RW_I^k$ $B_IW_R^k$	
4		$B_RW_R^k - B_IW_I^k$	$B_IW_R^k + B_RW_I^k$		
5	A	$A_R + B_RW_R^k - B_IW_I^k$	$A_I + B_IW_R^k + B_RW_I^k$		LOOP ON THESE STEPS
6	B	$A_I - B_IW_R^k - B_RW_I^k$	$A_R - B_RW_R^k + B_IW_I^k$		
7	A′			$B_RW_R^k$ $B_IW_I^k$ $B_RW_I^k$ $B_IW_R^k$	
8	B′	$B_RW_R^k - B_IW_I^k$	$B_IW_R^k + B_RtW_I^k$		

Fig 5—Data-bus utilization is 100% *in a 4-cycle butterfly. Here you can see that using just one multiplier doesn't hinder throughput.*

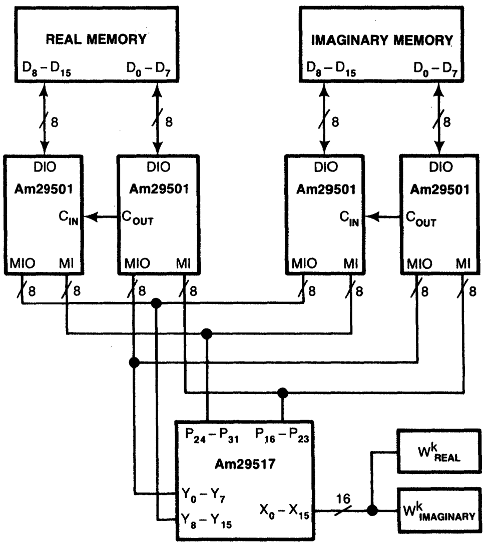


Fig 6—This FFT circuit *uses only one multiplier and a handful of other components.*

Fig 5's table shows how to achieve even higher performance. You can copy cycle 4 into cycle 8, which allows the processor to jump to cycle 5 and produce a 4-cycle loop. This action doubles the original throughput. In this case, the data bus experiences 100% utilization and the ALUs 75%, but the multipliers are still employed only 25% of the time. Clearly, you don't need four multipliers. In fact, you can achieve the same performance with only one multiplier by pipelining an additional butterfly. A design example demonstrates this technique.

Start a butterfly every four cycles

The **Fig 6** design uses a real-variable processor and an imaginary-variable processor, each with two Am29501s to provide 16-bit precision. (The Am29501 is an 8-bit, cascadable processor comprising an ALU, a set of six registers, and three data ports.) The two processors also share an Am29517 16-bit parallel multiplier, which has two 16-bit inputs, X and Y. The Y input connects to the multiplier I/O (MIO) port on the real and imaginary 29501s; the X input is driven either by a PROM containing the complex constants W^k or by Am29526/27/28/29 sine/cosine generators. The high-order 16 bits of the multiplier output ($P_{16\text{-}31}$) go to the 29501s' multiplier input (MI) ports, while the low-order 16 bits of the product are ignored. Memory consists of static RAM with a cycle time of less than 100 nsec.

The microcode needed to perform one butterfly is 10 cycles long **(Fig 7a)**, but you should note two things. First, registers are never used for more than four cycles, so the processor can load them with new values every four cycles. This, in turn, means that it can start a new butterfly every four cycles.

Second, you can superimpose each line of code onto the line four cycles below it without causing resource conflicts. For example, **Fig 7b's** code superimposes lines 1 through 4 over lines 5 through 8 to start computing a second butterfly while the first is still executing. This process repeats in **Fig 7c's** code, where lines 5 through 8 are then superimposed over lines 9 through 12. These last four lines contain the code necessary to compute three concurrent butterflies. You must ensure only that, when the processor reads or writes A or B, it knows exactly to which butterfly the data applies.

FFTs reduce DFT computations

Fourier transforms mathematically approximate a signal's transformation from the time domain to the frequency domain, and several algorithms implement the technique. All are based on the discrete Fourier transform (DFT), which sums time-domain samples (x(n)) that are multiplied by complex constants:

$$X(k) = \sum_{n=0}^{N-1} x(n)W^{2uk/N}$$
$$k = 0, 1, \ldots, N-1,$$

where $W=e^{-j\pi}$ and each X(k) is a frequency-domain Fourier coefficient. The computation of each coefficient requires N complex multiplications, where N is the number of samples. This results in N^2 complex multiplications.

The fast Fourier transform (FFT) reduces complex multiplications by eliminating redundant calculations, using the equation

$$X(k) = G(k) + W^{2k/N} H(k) \qquad k = 0, 1, \ldots, N-1, \qquad \textbf{(1)}$$

where G(k) is the DFT of the even samples in x(n), and H(k) is the DFT of the odd samples. (The algorithm discussed here is a radix-2 decimation-in-time algorithm; other schemes may provide additional benefits.)

Shaving points

Based on **Eq 1, Fig A** shows an 8-point DFT that's divided into two 4-point DFTs, one of which operates on even samples while the other operates on odd ones. As **Fig A** shows, the results are summed to produce the 8-point DFT result. This configuration takes advantage of the fact that G(k) and H(k) have period N/2. In other words,

$$G\left(k + \frac{N}{2}\right) = G(k)$$
$$H\left(k + \frac{N}{2}\right) = H(k).$$

Each 4-point DFT requires $N^2=16$ complex multiplications, and combining the intermediate results to obtain the eight frequency-domain coefficients requires one complex multiplication for each coefficient (the arrows represent multiplication by the noted constant). Thus, the **Fig A** transform requires a total of 40 (16+16+8) complex multiplications—a savings of 24 compared with the 64 multiplications required to compute an 8-point DFT directly.

By repeating this process and dividing the 4-point DFTs into 2-point DFTs, you can eliminate even more computations. The 8-point FFT represented in **Fig B** requires eight complex multiplications for the four 2-point DFTs plus 16 other complex multiplications, for a total of 24. In general, the number of complex multiplications equals the number of columns in the representation ($\log_2 N$) times the number of samples.

Another technique allows you to cut multiplications in half again. In **Fig C,** each circle represents a

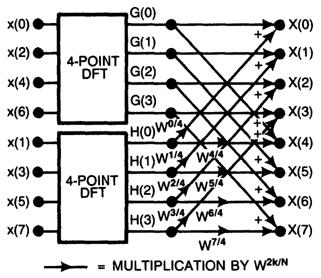


Fig A—Compared with direct computation of an 8-point DFT, *decomposing it into two 4-point DFTs saves 24 complex multiplications.*

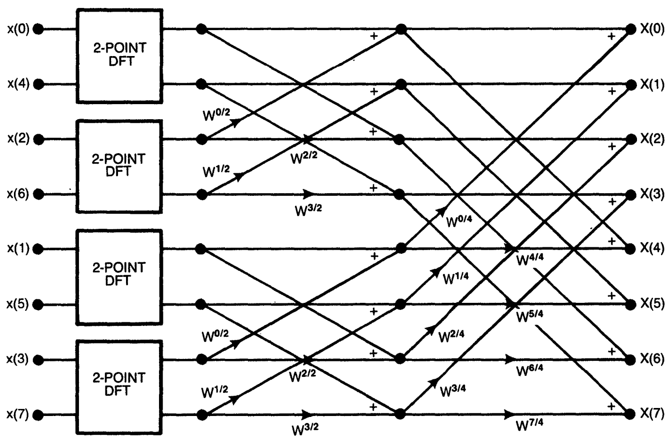


Fig B—This scheme, which uses four 2-point DFTs *to transform eight time-domain samples, requires a total of 24 complex multiplications: eight for the four 2-point DFTs plus the 16 represented here by arrows.*

sum and a difference. Using the structure in **Fig D,** you define A′ and B′ as follows:

$$A' = A + BW^{2k/N}$$
$$B' = A - BW^{2k/N}.$$

This notation results because W is a complex exponential and therefore periodic:

$$W^{2k/N} = -W^{2(k+\frac{N}{2})/N}.$$

Because you can use the product $B \times W^{2k/N}$ to calculate both A′ and B′, the total number of complex multiplications drops to $(N\log_2 N)/2$. This structure is the primitive operation in FFT calculations and is called a butterfly operation. Note that each circle in **Fig C** is a butterfly operation. This fact suggests a pipelined operation, optimized to execute butterflies, that can exploit the algorithm's highly repetitive nature.

Dissecting the butterfly

Each butterfly consists of two calculations:

$$A' = A + BW^k$$
$$B' = A - BW^k,$$

where A′, B′, A, B and W^k are complex numbers. (Here, the exponent 2k/N is consolidated into one term, k, for simplicity.) Dividing these into their real and imaginary parts yields

$$\begin{aligned} A' &= (A_R + j\,A_I) \\ &\quad + (B_R + j\,B_I)(W_R{}^k + j\,W_I{}^k) \\ B' &= (A_R + j\,A_I) \\ &\quad - (B_R + j\,B_I)(W_R{}^k + j\,W_I{}^k). \end{aligned}$$

Expanding the products gives

$$\begin{aligned} A' &= (A_R + j\,A_I) + (B_R\,W_R{}^k + \\ &\quad j\,B_I\,W_R{}^k + j\,B_R\,W_I{}^k - B_I\,W_I{}^k) \\ B' &= (A_R + j\,A_I) - (B_R\,W_R{}^k + \\ &\quad j\,B_I\,W_R{}^k + j\,B_R\,W_I{}^k - B_I\,W_I{}^k). \end{aligned}$$

Dividing A′ and B′ into their real

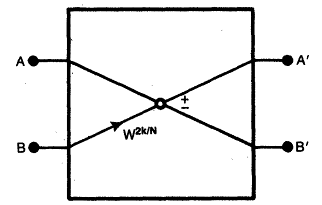


Fig D—The 2-point DFT, or butterfly, *is the primitive operation in FFT calculations.* ***Fig C*** *includes many such operations and thus lends itself to a pipelined implementation.*

and imaginary parts yields four equations:

$$\begin{aligned} A_R' &= A_R + B_R\,W_R{}^k - B_I\,W_I{}^k \\ B_R' &= A_R - B_R\,W_R{}^k + B_I\,W_I{}^k \\ A_I' &= A_I + B_I\,W_R{}^k + B_R\,W_I{}^k \\ B_I' &= A_I - B_I\,W_R{}^k - B_R\,W_I{}^k. \end{aligned}$$

These equations share common terms that need be calculated only once per butterfly. Regrouping terms yields additional savings:

$$\begin{aligned} A_R' &= A_R + (B_R\,W_R{}^k - B_I\,W_I{}^k) \\ B_R' &= A_R - (B_R\,W_R{}^k - B_I\,W_I{}^k) \\ A_I' &= A_I + (B_I\,W_R{}^k + B_R\,W_I{}^k) \\ B_I' &= A_I - (B_I\,W_R{}^k + B_R\,W_I{}^k). \end{aligned}$$

You can now determine the number of calculations necessary per butterfly: four multiplications to compute $B \times W^K$, a subtraction to calculate real A′ and B′, an addition to calculate imaginary A′ and B′, two final additions for A′, and two final subtractions for B′. This process yields a total of four products, three additions and three subtractions per butterfly.

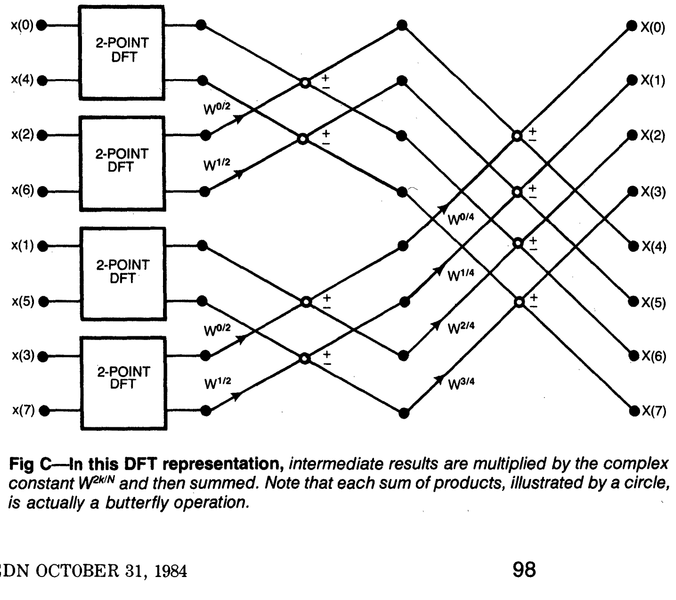


Fig C—In this DFT representation, *intermediate results are multiplied by the complex constant $W^{2k/N}$ and then summed. Note that each sum of products, illustrated by a circle, is actually a butterfly operation.*

(a)

$A' = A + BK$
$B' = A - BK$

		REAL							IMAGINARY							MULTIPLIER	
STEP	DIO	ALU	A_1	A_2	A_3	B_1	B_2	B_3	ALU	A_1	A_2	A_3	B_1	B_2	B_3	MIO	MULT
1	READ B					DI							DI				
2																B_R	
3	READ A		DI							DI							B_R W_R
4		$A_1 - MI$			MI		ALU									B_I	B_R W_I
5									$A_1 - MI$			MI		ALU			B_I W_R
6		$A_1 + A_3$		ALU					$B_2 - MI$					ALU	MI		B_I W_I
7		$B_2 + MI$				MI	ALU		$A_1 + B_3$		ALU						
8	WRITE B_2								$A_2 + A_3$		ALU						
9		$A_2 - B_1$		ALU													
10	WRITE A_2																
11																	
12																	
13																	
14																	
15																	
16																	
17																	
18																	

(b)

$A' = A + BK$
$B' = A - BK$

		REAL							IMAGINARY							MULTIPLIER	
STEP	DIO	ALU	A_1	A_2	A_3	B_1	B_2	B_3	ALU	A_1	A_2	A_3	B_1	B_2	B_3	MIO	MULT
1	READ B					DI							DI				
2																B_R	
3	READ A		DI							DI							B_R W_R
4		$A_1 - MI$			MI		ALU									B_I	B_R W_I
5	READ B					DI			$A_1 - MI$			MI	DI	ALU			B_I W_R
6		$A_1 + A_3$		ALU					$B_2 - MI$					ALU	MI	B_R	B_I W_I
7	READ A	$B_2 + MI$	DI			MI	ALU		$A_1 + B_3$	DI	ALU						B_R W_R
8	WRITE B_2	$A_1 - MI$			MI		ALU		$A_2 + A_3$		ALU					B_I	B_R W_I
9		$A_2 - B_1$		ALU													
10	WRITE A_2																
11																	
12																	
13																	
14																	
15																	
16																	
17																	
18																	

(c)

$A' = A + BK$
$B' = A - BK$

		REAL							IMAGINARY							MULTIPLIER	
STEP	DIO	ALU	A_1	A_2	A_3	B_1	B_2	B_3	ALU	A_1	A_2	A_3	B_1	B_2	B_3	MIO	MULT
1	READ B					DI							DI				
2																B_R	
3	READ A		DI							DI							B_R W_R
4		$A_1 - MI$			MI		ALU									B_I	B_R W_I
5	READ B					DI			$A_1 - MI$			MI	DI	ALU			B_I W_R
6		$A_1 + A_3$		ALU					$B_2 - MI$					ALU	MI	B_R	B_I W_I
7	READ A	$B_2 + MI$	DI			MI	ALU		$A_1 + B_3$	DI	ALU						B_R W_R
8	WRITE B_2	$A_1 - MI$			MI		ALU		$A_2 + A_3$		ALU					B_I	B_R W_I
9	READ B	$A_2 - B_1$		ALU		DI			$A_1 - MI$			MI	DI	ALU			B_I W_R
10	WRITE A_2	$A_1 + A_3$		ALU					$B_2 - MI$					ALU		B_R	B_I W_I
11	READ A	$B_2 + MI$	DI			MI	ALU		$A_1 + B_3$	DI	ALU						B_R W_R
12	WRITE B_2	$A_1 - MI$			MI		ALU		$A_2 + A_3$		ALU					B_I	B_R W_I
13																	
14																	
15																	
16																	
17																	
18																	

Fig 7—The microcode needed to execute one butterfly *in Fig 6's circuit is 10 cycles long* ***(a)****. By following the code in* ***(b)*** *you can start computing a second butterfly concurrently. The code in* ***(c)*** *starts a third concurrent butterfly.*

Pipeline the FFT processor to reduce bus's 50% idle time

To help keep these values straight, the circuit uses the Am29520 multilevel pipeline register. **Fig 8** depicts how this device operates in the address pathway to memory. Every four clock cycles, at the beginning of a new butterfly, the Am29540 FFT address sequencer generates a new set of addresses for A and B. The 29520s store these addresses temporarily in internal registers that are configured as a 4-deep pipeline. As each new address is clocked into the first pipeline register, previously stored addresses advance to the next register. You can select any register for output and access the appropriate address for any microcode cycle. **Fig 9** illustrates the order in which this design stores and retrieves data addresses.

The 29540 also generates addresses for the PROM containing W^k. It creates a new address for each butterfly and then stores it in an external register. Because complex products are computed on successive cycles, the address to the PROM changes at the beginning of each new butterfly (that is, every four cycles).

Microcode lines 9 through 12 then execute as a loop until the 29540's FFT Complete signal goes active. The entire transform requires only 12 words of microcode: The first eight preload the pipeline, while the last four perform the computations.

Word size can almost triple

In these FFT implementations, you must be concerned about word growth. Because the FFT butterfly produces outputs by adding terms, butterfly outputs may require more bits than each input has. Specifically, consider the equation $A'=A+BW^k$, where A', A, B and W^k are complex. This equation, one of the butterfly's two basic calculations, represents vector rotation and addition. The term BW^k merely describes a rotation of vector B by unit vector W^k; the result is added to vector A. The magnitude of A' can therefore be twice as large as A or B.

Unfortunately, this problem is more insidious than it appears. Although complex magnitudes do no more than double at each stage, the real and imaginary components of these complex values can increase by more than that amount. Indeed, they can increase by $1+\sqrt{2}$, or 2.41, for decimation-in-time algorithms, which is the type used here. They can even increase by $2\times\sqrt{2}$, or 2.82, for decimation-in-frequency algorithms, which use a different butterfly technique (see **reference**). In either case, you must allow for as much as two bits of growth in every stage. You could design a system with sufficient extra growth bits, but this approach is wasteful and expensive, particularly if the transform has many stages.

An inexpensive alternative is to use the block-floating-point scheme. This technique uses a common block

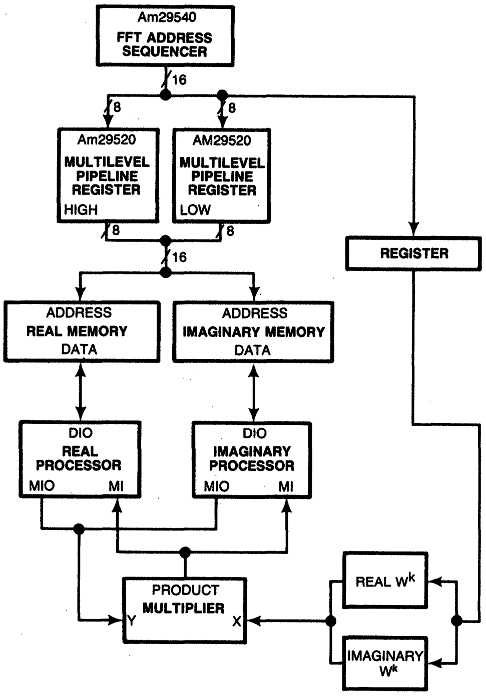


Fig 8—The system must keep track of many sets of data *when computing concurrent butterflies. To do so, it must incorporate an FFT address sequencer.*

			29520 CONTENTS				
CYCLE	MEMORY OPERATION	29520 INPUT	REG A_1	REG A_2	REG B_1	REG B_2	29520 OUTPUT
0		$*B_0$					
1	READ B_0		$*B_0$				$*B_0$
2		$*A_0$	$*B_0$				
3	READ A_0		$*A_0$	$*B_0$			$*A_0$
4		$*B_1$	$*A_0$	$*B_0$			
5	READ B_1		$*B_1$	$*A_0$	$*B_0$		$*B_1$
6		$*A_1$	$*B_1$	$*A_0$	$*B_0$		
7	READ A_1		$*A_1$	$*B_1$	$*A_0$	$*B_0$	$*A_1$
8	WRITE B_0	$*B_2$	$*A_1$	$*B_1$	$*A_0$	$*B_0$	$*B_0$
9	READ B_2		$*B_2$	$*A_1$	$*B_1$	$*A_0$	$*B_2$
10	WRITE A_0	$*A_2$	$*B_2$	$*A_1$	$*B_1$	$*A_0$	$*A_0$
11	READ A_2		$*A_2$	$*B_2$	$*A_1$	$*B_1$	$*A_2$
12	WRITE B_1	$*B_3$	$*A_2$	$*B_2$	$*A_1$	$*B_1$	$*B_1$

*indicates "address of"

Fig 9—Four registers in the address sequencer *prove sufficient to store the various data addresses needed to compute three concurrent butterflies.*

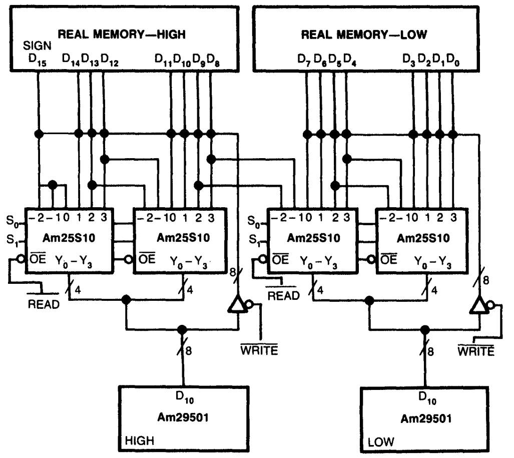

REAL MEMORY—HIGH
SIGN
D_{15} $D_{14}D_{13}D_{12}$ $D_{11}D_{10}D_{9}D_{8}$
REAL MEMORY—LOW
$D_7D_6D_5D_4$ $D_3D_2D_1D_0$
-2-1 0 1 2 3
Am25S10
$\overline{OE}$ Y_0-Y_3
S_0
S_1
READ
WRITE
4
8
D_{10}
Am29501
HIGH
LOW

Fig 10—To avoid data overflow caused by word growth, *implement a block-floating-point technique with 4-bit shifters inserted into the data-read pathway.*

exponent for all data. If the system expects or detects an overflow, it shifts data to the right and increments the block exponent.

The circuit shown in **Fig 10** implements this approach with two Am25S10 4-bit shifters inserted between memory and the real and imaginary processors in the data-read pathway. The shifters allow you to divide data read from memory by 1, 2 or 4. Each time the system writes data to memory, external logic compares the two high-order data bits to the sign bit.

If the high-order bits differ from the sign bit, the data's magnitude has expanded into the high-order bits, and an overflow could occur in the next column of butterflies because data could increase by 2.41. Consequently, if logic detects an expansion into the high-order bits, it sets a flag. Then, when the next column begins (signaled by Iteration Complete from the 29540), the system reads all data as shifted to the right by zero bits (if no expansion took place), by one bit (if the expansion occurred only in D_{13}) or by two bits (if the expansion occurred in D_{14}). Note that the sign bit must be duplicated in the high-order bits. Upon receipt of Iteration Complete, the block-exponent counter increments by 0, 1 or 2. The host CPU can then read this value to determine the Fourier coefficients' absolute magnitude.

Though the 29540 FFT address sequencer has many operating modes to accommodate varying architectures and algorithms, the system described here executes a radix-2 decimation-in-time transform that doesn't produce data in a convenient sequence. You'd like data for the first frequency notch to occupy the lowest memory location, data for the second notch to occupy the next lowest location, and so on. To remedy this situation, you can either scramble data points before they enter the algorithm so that they emerge in the proper sequence, or you can scramble them afterwards.

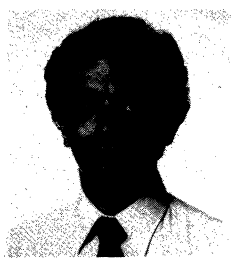
Although this article's architecture describes a special-purpose FFT processor, you can use it as a general-purpose signal processor. Many signal-processing algorithms have a sum-of-products notation that is well suited to this design. Essentially, you can substitute the PROM that contains W^k with a RAM that the host processor loads. In this way, you can easily implement windowing and scaling operations. **EDN**

Reference

Oppenheim, Alan V and Schafer, Ronald W, *Digital Signal Processing,* Prentice-Hall, Englewood Cliffs, NJ, 1975.

Authors' biographies

Robert Cohen worked as a design engineer in product planning at Advanced Micro Devices (Sunnyvale, CA) from 1981 to 1984. He is now a private consultant. He received a BSE degree in computer science and engineering from the University of Pennsylvania in 1981. His favorite food is a combination of guacamole and knishes.



Robert Perlman is a senior product planning engineer with the DSP/array processing group at Advanced Micro Devices. He holds a BSEE degree from the Rensselaer Polytechnic Institute and an MSEE degree from Johns Hopkins University, and he has done design work in airborne digital design processing for Westinghouse.

VERY HIGH SPEED FLOATING POINT PROCESSOR.

B.J.New.

Advanced Micro Devices, Sunnyvale, California / USA.

Introduction:
In the past many computers have been augmented with floating point accelerators. This paper describes a single-chip floating point processor, which provides add, subtract and multiply capabilities at very high speed. Utilizing ECL technology, the functions are performed in a single cycle of approximately 100ns.

Arithmetically intensive problems requiring the use of floating point accelerators may be found in many areas. In particular, graphics, robotics and digital signal processing are intensive in vector and matrix arithmetic. The floating point processor described is suited to such operations, having internal pathing to support efficient computation of sums of products.

The Am29325 Floating Point Processor:
The Am29325 /1/ is a single-precision floating point processor which may be operated either in the IEEE (IEC) format, or in the DEC format. Figure 1 is a block diagram of the processor. The heart of the processor is a block of combinatorial logic which contains no state information whatsoever. The instructions which may be performed in this logic are detailed in Table 1. The arithmetic instructions are add, subtract, multiply and a 2-minus instruction.

The purpose of the 2-minus instruction is to support division. It was not possible to include division in the processor, and it is therefore suggested that the Newton-Raphson algorithm be used. Initial estimates of the reciprocal may be generated from a look-up table. Depending on the size of this look-up table, one or more iterations of the Newton-Raphson algorithm may be performed to refine the estimate to the desired accuracy. Other than the initial estimate, the 2-minus instruction allows the iteration to be performed in 3 cycles, without the insertion of constants. Including the loading of the operands and the final multiplication, the division may be performed in 3N+3 cycles, where N is the number of Newton-Raphson iterations.

Besides the arithmetic instructions, four conversion instructions are provided. These allow for conversions between fixed and floating point, and between the IEEE and DEC formats. A pin is provided to determine in which format the operands are assumed to be for the purposes of computation. Four rounding modes are provided; round to even, to positive infinity, to negative infinity and to zero. Two infinity modes are available, projective and affine.

Six flags are generated to indicate the status of an operation. These are:

INVALID - Indicates that the combination of operands do not allow the operation to be performed.

OVERFLOW - The result is too large for the format.

UNDERFLOW - The result is too small for the format.

INEXACT - The rounded result does not equal the infinitely precise result.

ZERO - The result is zero.

NAN - The operation resulted in a Not-a-number.

For ease of use, the flags are unencoded, and are registered such that they relate to data in the result register.

Two operand registers, R and S, are provided together with a result register, F. These allow pipelining of data onto, and off the chip. However, for maximum flexibility, the input pipeline stage and the output pipeline stage may independently be made transparent. This permits double-pipelined, single-pipelined (at input or output) or unpipelined operation. This last mode of operation makes the part unique among integrated floating point processors, in that, combined with the two 32-bit input busses and the 32-bit output bus, it allows completely asynchronous fall-through operation.

Internal feedback paths are provided within the part, which allow a result to be loaded directly into the R register, independent of the operation of the F register, and which also allow the F register contents to be used as an operand, in place of the S register contents. Each register is independently enabled, and may therefore be used to retain intermediate values. This facility is particularly useful in computing Newton-Raphson iterations, and in summing products as described below.

An Array Processor using the Am29325:
Figure 2 is a block diagram of a small array processor using the Am29325. A high-speed multi-port memory is used to provide storage for operands such that they may be accessed in simultaneous pairs. These operands may originate in the data memory, or may be intermediate results from the processor. One of these operands may be replaced with a value drawn from a non-volatile coefficient store.

The array processor is microprogram controlled, with memory addresses being derived directly from the microcode. This is probably inefficient for large programs, and some form of microprogrammed address generator would need to be added. The interface to the host processor is deliberately undefined, as this is user dependent.

As a benchmark, this processor can perform FFT butterflies in the canonical time of 1Ø cycles. At a 1ØØns cycle time, this permits one butterfly every lus, or a 1024-pt complex transform in 5.12ms. A simple modification to the architecture allows a second Am29325 to be incorporated to give a complex arithmetic processor. This doubles the throughput for the FFT, reducing the computation for the 1024-pt transform to 2.56ms.

Because many floating point applications essentially comprise vector and matrix arithmetic, its performance in this area is of particular interest. Figure 3 illustrates the operation of the part when summing products, the kernel operation of matrix arithmetic. To initiate the process, two operands are loaded into the R and S registers (Figure 3a). These are multiplied, and the product entered into the F register, initiating the accumulator (Figure 3b). Simulaneously, two new operands are loaded, starting the first iteration of the inner loop. These are again multiplied, but the product is entered into the R register, leaving the accumulated value in the F register unchanged (Figure 3c). Now the temporary value in the S register is added to the accumulated value in the F register, and the new accumulated value is entered back into the F register (Figure 3d). During this cycle the next pair of operands is also loaded into the R and S registers, starting the next iteration of the inner loop.

These last two steps described (Figures 3c and 3d) constitute the inner loop, allowing vector dot products to be computed at a rate of two cycles per point. Typical of an operation performed in a graphics transformation, a 4-element vector may be multiplied by a 4x4 matrix in 32 cycles. At a 10MHz clock rate, this gives a throughput of 3.2us per vertex transformation.

While the Am29325 only provides single-precision floating point operation, the Am29300 family also provides bus-compatible devices which may be used to enhance the capabilities of the array processor described. The Am29332 Integer Processor offers a wide range of arithmetic, logic and shift facilities. This device may be operated with a reduced width data path, allowing words of 1 to 4 bytes. The internal architecture is designed for efficient programming of floating point operations, and may therefore be used to support the Am29325 with double-precision operations. To assist in double-precision floating point multiplication, or for integer multiplication, the Am29323 32-bit Multiplier provides 32x32-bit multiplication in a single cycle, and has internal facilities for multi-cycle expansion to 128x128.

These additional arithmetic elements have the same 32-bit, 3-bus architecture as the Am29325. This allows them to be added in parallel. The routing of operands to the appropriate arithmetic element is a simple microcode task.

Conclusion:

This paper has described an integrated floating point processor, which requires only the addition of memory elements and a microprogram sequencer to implement a high-performance array processor, with 10Mflop single-precision throughput. This array processor may be expanded to include further floating point units to improve throughput, or other arithmetic devices to increase functionality.

References:

/1/ - ,"Am29325 Floating Point Processor Data Sheet," Advanced Micro Devices, Sunnyvale, California, USA, 1984.

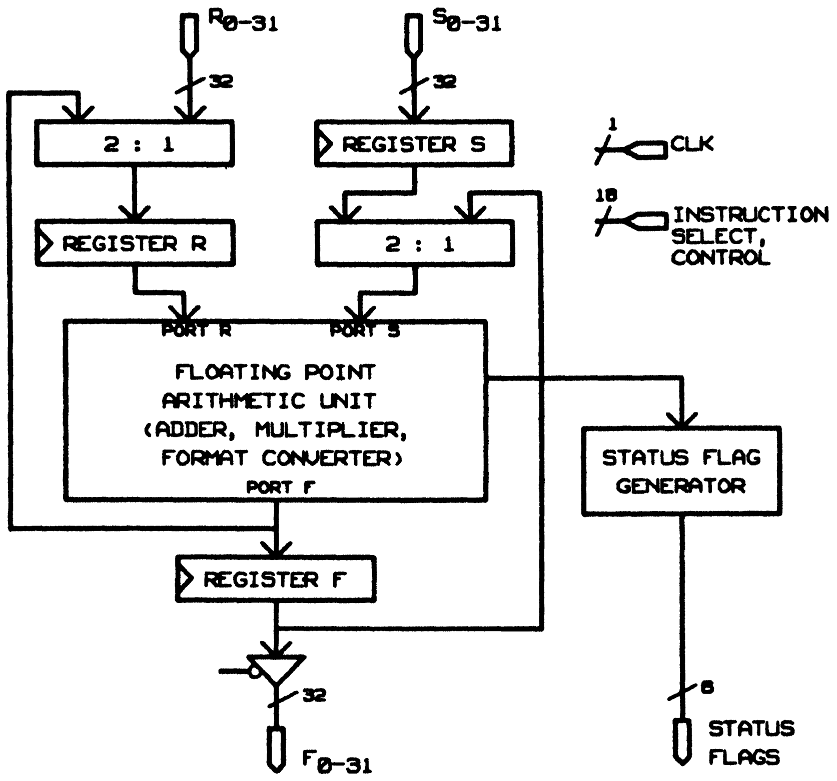


Figure 1: Am29325 Block Diagram.

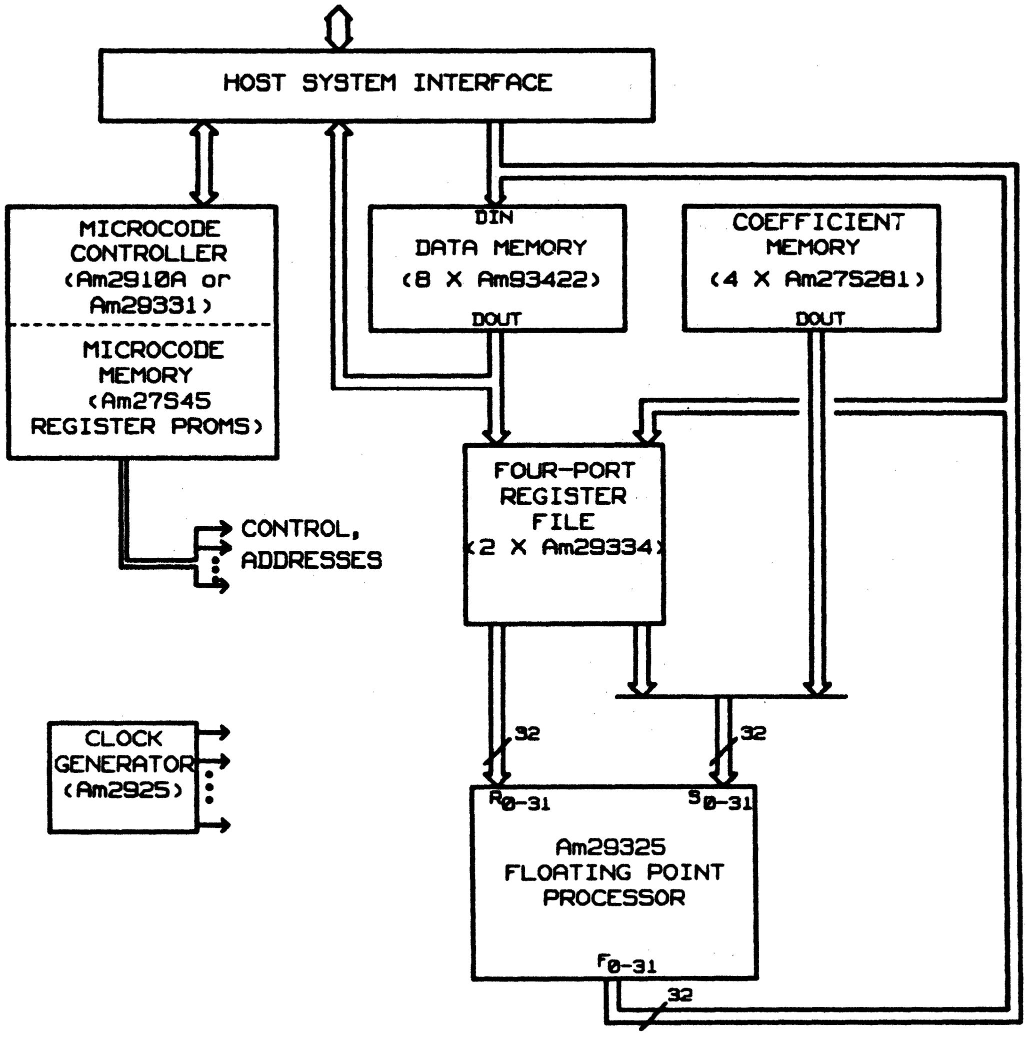

HOST SYSTEM INTERFACE
MICROCODE CONTROLLER (Am2910A or Am29331)
MICROCODE MEMORY (Am27S45 REGISTER PROMS)
DIN
DATA MEMORY (8 X Am93422)
DOUT
COEFFICIENT MEMORY (4 X Am27S281)
DOUT
FOUR-PORT REGISTER FILE (2 X Am29334)
CONTROL, ADDRESSES
CLOCK GENERATOR (Am2925)
32
32
R0-31
S0-31
Am29325 FLOATING POINT PROCESSOR
F0-31
32

Figure 2: Array Processor Block Diagram.

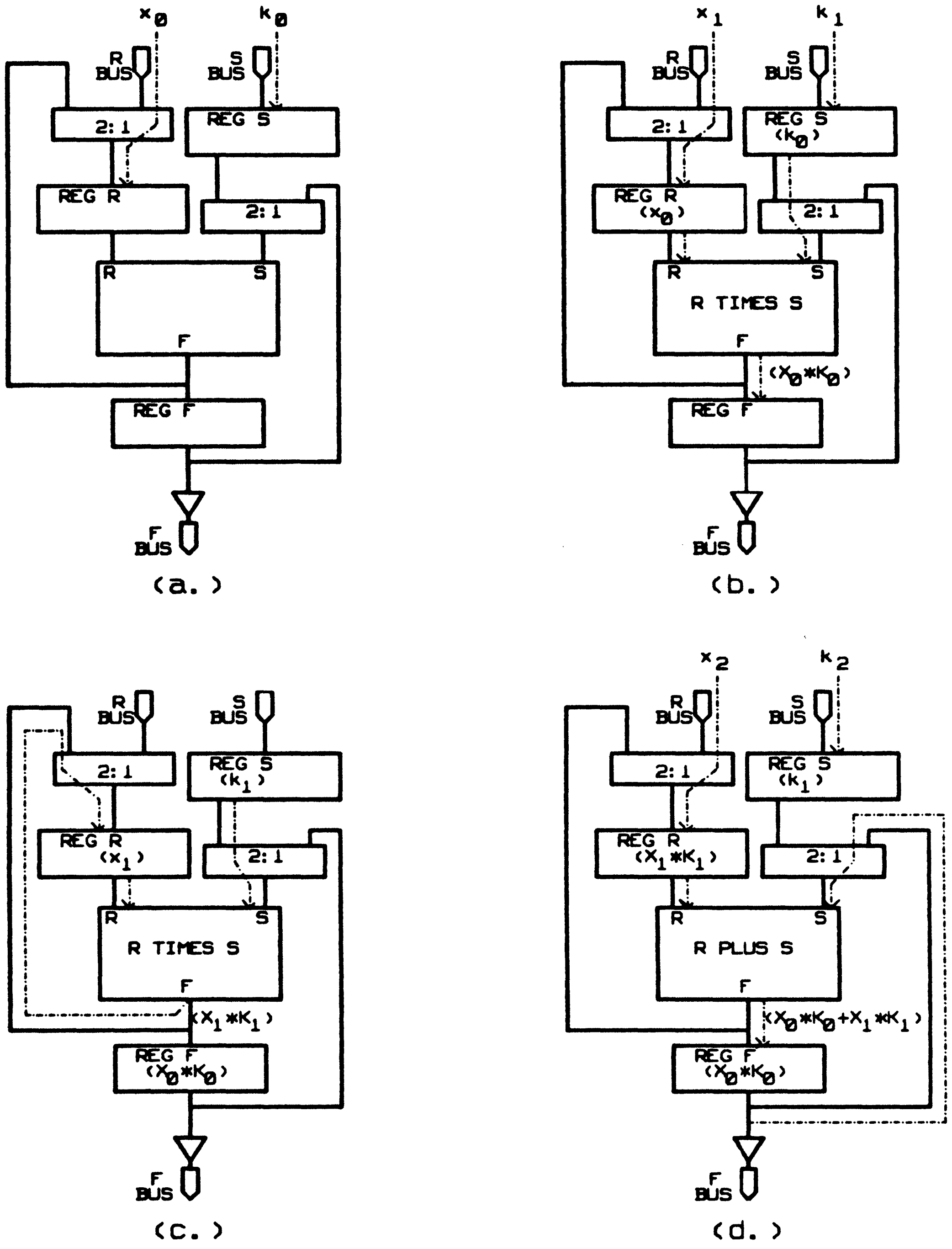


Figure 3: Multiply-and-Accumulate Operation.

MNEMONIC	OPERATION
R PLUS S	Add floating point operands R and S
R MINUS S	Subtract floating point operand S from floating point operand R
R TIMES S	Multiply floating point operands R and S
2 MINUS S	Constant floating point subtraction for Newton-Raphson division (see text)
INT-TO-FP	Convert floating point operand R to integer
FP-TO-INT	Convert integer operand R to floating point
IEEE-TO-DEC	convert IEEE floating point operand R to DEC floating point format
DEC-TO-IEEE	convert DEC floating point operand R to IEEE floating point format

Table 1: Am29325 Instruction Set.

TECHNOLOGY EVOLUTION TOWARDS DIGITAL CUSTOMER ACCESS

Hadi Ibrahim
Advanced Micro Devices
901 Thompson Place
Sunnyvale, CA 94088

ABSTRACT

Integrated Circuits providing a system solution for PBX analog and digital subscriber line cards are described. The paper focusses on the need for compatibility at the systems interface for successful transition to the Integrated Services Digital Network (ISDN).

INTRODUCTION

At present, the subscriber lines handle analog signals providing voice transmission for telephony and data transmission using voice-band modems. Within the public network and much more rapidly in the private network, a transition towards digital subscriber lines is occurring. This will allow simultaneous transmission of voice and data and lead to the ISDN.

The line card in the PBX forms the interface between the subscriber line, the switching system and the control system. For analog subscribers, the line interface consists of circuits which digitize and process a single voice channel at the line card. The digital signal is then encoded according to u-law or A-law techniques and transferred to the switching system via a 64kbs timeslot on PCM highways. A microprocessor handles the routine signaling procedures associated with telephony and passes the information to the control system for further call processing.

The ISDN subscriber can generate two 64kbs voice/data channels and a 16kbs signaling channel. Voice is now digitized at the subscribers terminal and transmitted over one of the 64kbs channels. The digital line card contains transceivers to achieve transmission at low bit error rates. A microprocessor handles the signaling protocols within the 16kbs channel. Two voice/data channels are now required to be transferred to the switching system via 64kbs timeslots on PCM highways. Optionally, the signaling channel may also be transferred to the PCM highways at a 16kbs rate (up to four channels occupying a 64kbs timeslot).

End to end digital connections for the local ISDN subscribers of the PBX are completed through the switching system which operates on the 64kbs timeslots. If a digital carrier interface is available at the PBX, selective end to end digital connections are possible to subscribers on other digital exchanges.

Optimum usage of current bipolar and MOS device technologies has already resulted in the availability of integrated circuits for the analog line card. An evolutionary development of the digital line card requires that design of new integrated circuits is based on intelligent systems considerations. The transition to ISDN should have minimal impact on the PBX.

ANALOG SUBSCRIBER INTERFACE

CCITT[1] and various telecommunication administrations have developed standards for analog line card interfaces to ensure the desired quality of service and compatibility between networks and subscriber equipment. The standards have placed stringent requirements on the design of the integrated circuits.

A functional block design of a subscriber line system based on two LSI devices, the Subscriber Line Interface Circuit (SLIC) and the Subscriber Line Audio-processing Circuit (SLAC) is shown in Figure 1. The SLIC/SLAC combination fully integrates the BORSCHT functions, battery feed, overload protection, ringing, supervision, codec-filter, hybrid and testing.

Utilizing a high voltage bipolar technology, the SLIC feeds battery voltage (approximately -50V) to the subscriber line to allow remote powering of the telephone instrument. A switching regulator scheme minimizes the total system and IC power dissipation. The ring and test networks access the line through relays which are independently controlled by the SLIC. The device also contains circuits to monitor off-hook/on-hook status and off-hook during ringing. Two- to four-wire conversion occurs in the hybrid section and the voice signals are presented to the SLAC.

The SLAC uses advances in digital signal processing and MOS/LSI technologies to implement the codec filter and many other system functions. The user is able to digitally program balance networks, two-wire impedance levels and the gains in the transmit and receive paths.

Both the SLIC and SLAC interface easily to a microprocessor on the line card. Under software control, the microprocessor sets the characteristics of each line interface and processes the basic signaling functions. The subscriber lines are monitored at a fixed scan rate to detect changes in hook condition. The microprocessor uses this information to determine the subscribers current status and count pulses during the dialing phase. During the ringing phase, detection of off-hook signals that ring trip has occurred.

The presence of a microprocessor for control in the analog line card has an advantage in the migration to ISDN. Although the signaling procedures in the digital line card are more complex and in a different format, they carry the same basic functions. Thus software control and modifications will retain compatibility at the systems level.

DIGITAL SUBSCRIBER INTERFACE

The CCITT is in the process of defining a limited set of user-network interfaces for digital customer access to the ISDN. These interfaces will support a wide range of voice and data services using digital transmission over the subscriber line. The basic access to the subscriber is over two twisted pair circuits designated the "S" reference point[2]. The effective full duplex transmission rate at this interface is 144kbs. Each frame of information contains two 64kbs channels (B1 and B2) and a 16kbs channel (D). The B-channels carry digitized voice and data signals. The D-channel contains signaling information (control for B-channels) and optionally packet switched data. The signaling protocol used on the D-channel is LAPD which is derived from procedures of the CCITT X.25 recommendation.

A functional block design of the digital subscriber line system based on two devices, the Exchange Power Controller (EPC) and the Digital Exchange Controller (DEC) is shown in Figure 2. The functions on the digital line card are simpler since the digitization and processing of the voice channel now occurs at the subscribers terminal. A CMOS VLSI circuit, the Digital Subscriber Controller[3] (DSC) is being developed for implementing the functions of ISDN terminals. In addition to the codec-filter, the DSC contains the "S" transceiver, protocol processor and interfaces to a microprocessor.

The EPC uses a high voltage bipolar technology to feed a regulated -40V to the subscriber. Remote power feed is optional at the "S" interface but terminal designs may be significantly simplified if a minimum set of voice/data functions can be powered directly from the PBX. The device contains circuits for line supervision to aid maintenance and test functions. A simple microprocessor interface controls its modes of operation.

The DEC contains the transceiver which along with the transformer forms the electrical interface to the "S" reference point. Signaling is now completely digital and contained within the D-channel. The LAPD protocol is partially processed in the DEC and the resulting data may be accessed by the microprocessor. The microprocessor interface is interrupt driven and provides fast and efficient transfer of D-channel data.

SYSTEM INTERFACE

The switching system interface is common to both analog and digital line cards. It consists of PCM highways carrying in 64kbs timeslots and operating at multiples of 1.544MHz or 2.048MHz speeds.

Examples of 8-channel analog and digital subscriber line cards are shown in Figure 3 and 4 respectively. Both the SLAC and DEC contain circuits to implement a dual PCM interface with independent control of the transmit and receive paths. The availability of two PCM systems improves reliability and simplifies switch design for small systems. The frame sync signal identifies the beginning of a frame and all timeslots are referenced to it. The SLAC contains user progammable timeslot registers which determine the transfer of a single voice channel onto the PCM highways. Whereas, the DEC contains additional user programmable timeslot registers to transfer two 64kbs. B-channels and the 16kbs D-channel (optional) onto the PCM highways during each frame.

References

1. International Telephone and Telegraph Consultative Committee (CCITT) Recommendations G.711 and G.712, June 1980.
2. International Telephone and Telegraph Consultative Committee (CCITT) Draft Recoammendation I.431, June 1983.
3. H. Najafi, "Techniques for Implementation of ISDN Terminals," ICC-84, May 14, 1982.

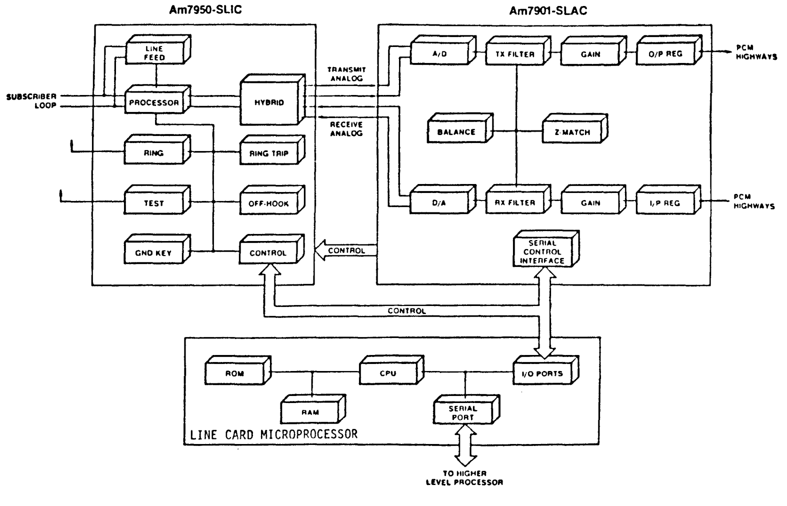


FIGURE 1: ANALOG SUBSCRIBER INTERFACES

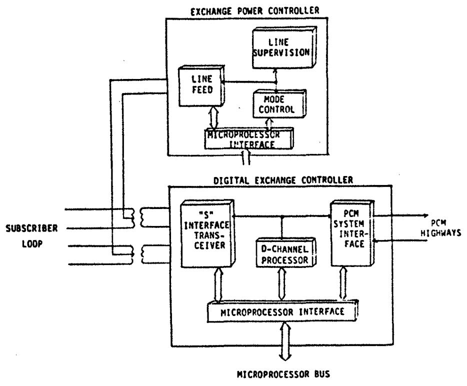


FIGURE 2: DIGITAL SUBSCRIBER INTERFACES

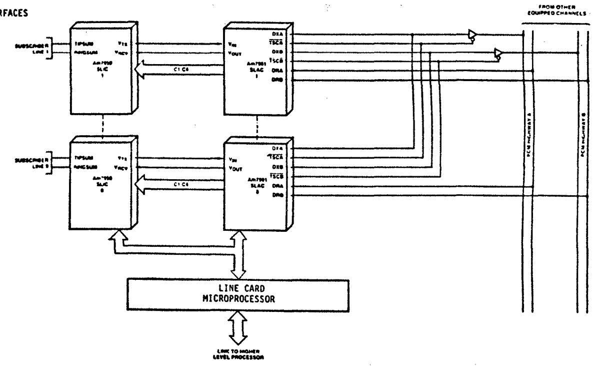


FIGURE 3: ANALOG SUBSCRIBER LINE CARD

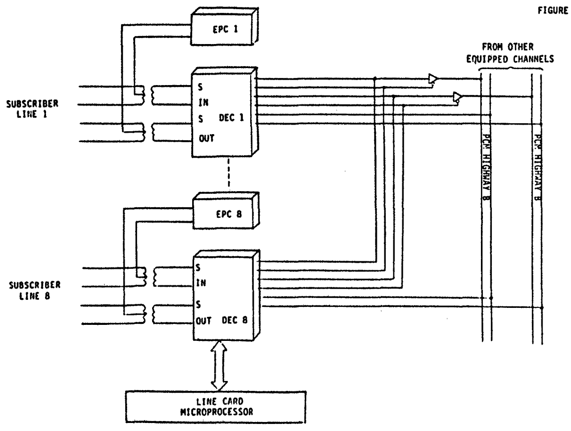


FIGURE 4 DIGITAL SUBSCRIBER LINE CARD

Stevan Eidson, Advanced Micro Devices, Sunnyvale, Calif.

Videotex hardware heralds another node—the home

Cottage industry, home shopping, and electronic mail may soon be as simple as a carriage return. New standards and chips will make such headquarters possible.

Within a few years, it probably will be routine to work, shop, send and receive mail, and even secure a home from fire and theft via the family videotex terminal. To that end, many networks designed for such activities are in place, and international message-handling standards are close to being rubber-stamped (see DATA COMMUNICATIONS, p. 159). Still missing from this scenario is the availability of low-cost terminals with firmware specifically designed to handle the varied—and occasionally cumbersome—methods used by existing videotex networks to present information.

To appreciate the problems involved in developing such a terminal, a brief discussion about the origins and principles of videotex is in order. This relatively new technology permits the exchange of information (text and graphics) over a standard communications medium—broadcast, telephone, and cable television (CATV)—with remote sites having some form of visual display and a response device.

Teletext, a poor cousin of videotex, involves a one-way information exchange. In the past, most teletext services have been tied to television broadcast networks, whereas many videotex services use the public telephone network or, more recently, CATV as a medium (Fig. 1).

Both videotex and teletext were developed in Britain in the early 1970s. Ceefax and Oracle, two of the original teletext services available in the U. K., were connected to more than 800,000 British television sets and are still quite popular. By 1979, the telecommunications portion of the U. K. Post Office, now British Telecom, had initiated a videotex service known as Prestel. Other protocols soon followed in France (Antiope), Japan (CAPTAIN), Canada (Telidon), and the United States (PLP). The French, Canadian, and American protocols will be discussed below.

The three components involved in a videotex terminal are the display and its controller, the user input device, and a modem that allows the terminal to talk over the transmission medium (Fig. 2). The display is typically a cathode ray tube (CRT), which requires a controller to translate the information presented to the terminal into the proper CRT voltage levels to create visual images. These images are formed from dots on a raster scan CRT. Thus the CRT's electron gun paints the screen dot-by-dot to display the appropriate picture or text.

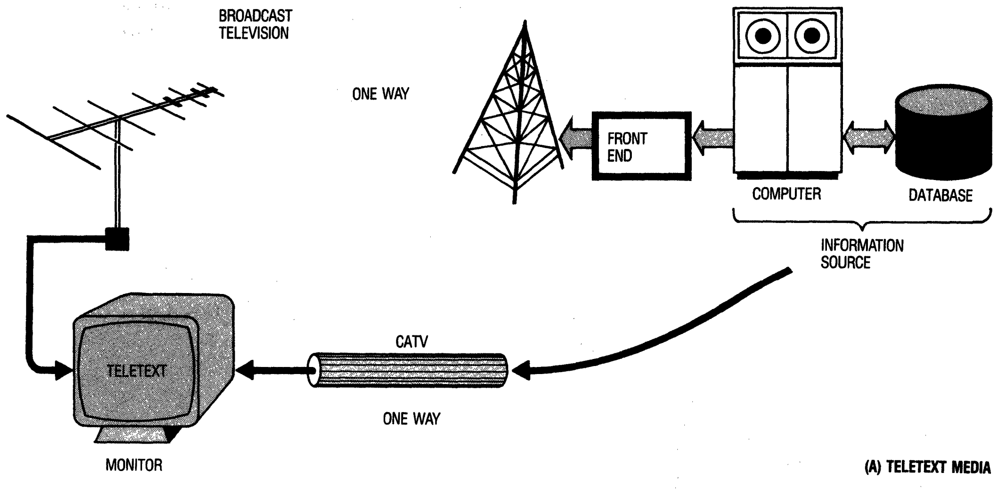
The input device is a standard keyboard, but mouses, touch screens, joysticks, speech decoders, and digitizing pads are often-used human interfaces. In fact, they are likely to become more common in videotex environments because the data generated by each can be encoded by the terminal to provide the same information. Input to the videotex terminal can be considered device-independent from the mechanism used to create the response.

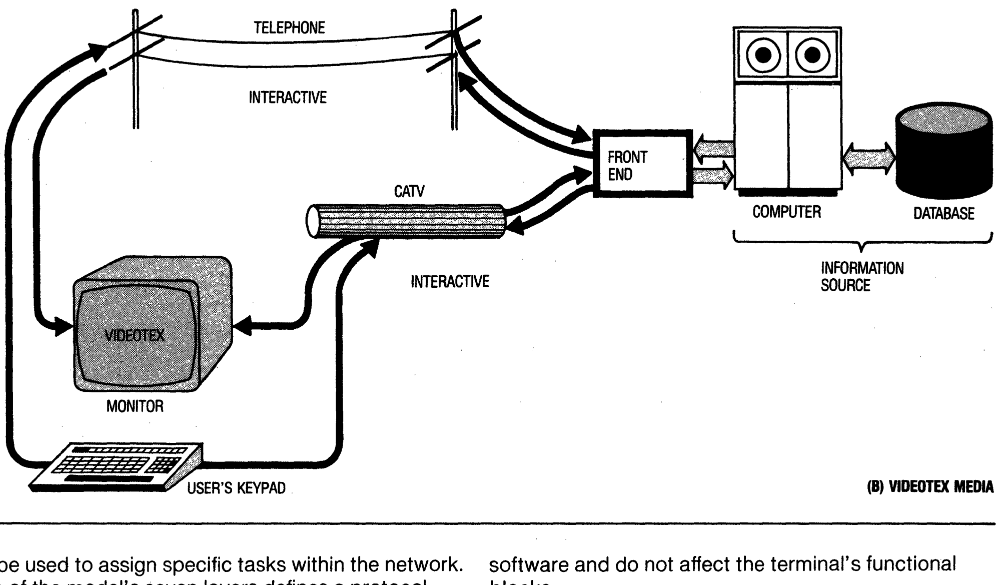
The most commonly used data transmission medium is the public-switched telephone network (PSTN). Data communications over the PSTN is typically accomplished by utilizing a modem, which converts analog (voice) data to digital data and vice versa. Until now, most videotex terminals have used standard full-duplex modem transmission techniques for data communications.

Key layers

To understand the concepts behind each videotex network, the Open Systems Interconnection of the International Standards Organization model should be mentioned (see "The OSI of the ISO"). Briefly, the OSI model gives the network designer an architecture that

1. The media. *Developed before videotex, teletext involves only a one-way flow of information. Teletext commonly uses TV broadcast as a transmission media, although CATV is also used. Videotex involves two-way interaction with the information source, and uses either public telephone or CATV as transmission media.*



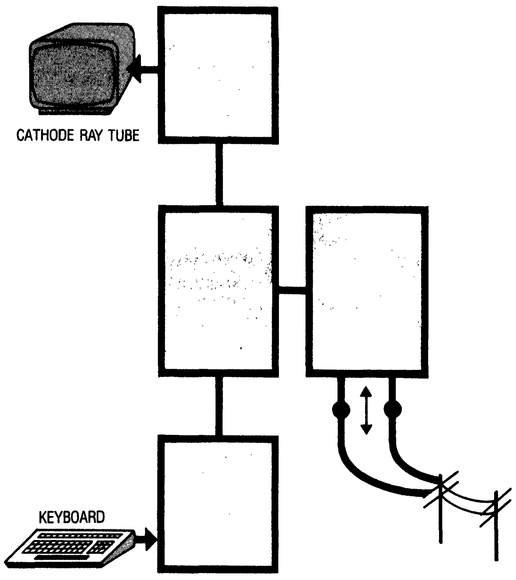


can be used to assign specific tasks within the network. Each of the model's seven layers defines a protocol adhered to by devices within the network.

As one goes from the human interface protocol at layer 7 to the physical network connection at layer 1, the information passed not only becomes less intelligible to users, but contains more error-checking or message-packet-ordering mechanisms. Of interest to a designer of videotex terminal hardware are level 6, the Presentation Layer, and level 1, the Physical Layer. In general, protocol layers 2 through 5 are implemented in software and do not affect the terminal's functional blocks.

The model's Physical Layer is concerned with the terminal's ability to send data across the transmission medium. This is done through a modem typically connected to the terminal via a common EIA- or CCITT-specified interface—usually EIA RS-232 or CCITT V.24. The Presentation Layer is involved with encoding user information into packages of bits. Two of the most familiar presentation types are American Standard Code for Information Interchange (ASCII) and Ex-

2. The box. *The terminal has four components: the keyboard interface, terminal controller, communications controller, and CRT controller.*



tended Binary Coded Decimal Information Code (EBCDIC). In these codes, bytes of data represent the characters of the alphabet, numbers, special textual symbols, and characters for the control of peripheral devices. These coding schemes were intended to be used with standard textual terminals and peripherals. Since a videotex terminal displays graphics as well as text, presentation layer protocols for representing graphics were developed as well. Although many presentation-level protocol standards exist, the trend is clearly toward a worldwide unified graphics standard for the Presentation Layer.

Unlike the Presentation Layer, most standards for data communications at the physical level have been divided between Europe and the United States. In Europe, modems specified by the Consultative Committee for International Telephone and Telegraph (CCITT) are accepted as standard; in North America, modems have been specified mostly by AT&T, with CCITT specifications used occasionally.

Current videotex terminals use 300-bit/s full-duplex modems and 1,200-bit/s half-duplex modems. The CCITT specification for 300-bit/s full-duplex modems is V.21. The corresponding North American 300-bit/s standard is the Bell 103 specification.

The 1,200-bit/s half-duplex modem standard in Europe is CCITT V.23. Because it is inconvenient to transmit in only one direction at a time (half-duplex), a low bit-rate backward channel is defined for V.23 at 75 bit/s. The Bell 202 is the North American 1,200-bit/s half-duplex modem specification; like V.23, this standard also specifies a low bit-rate backward channel at 5 bit/s. However, for videotex applications, the 5-bit/s backward channel is insufficient to transmit at reasonable speeds from the terminal to the central computer or information source. To correct this problem, the Canadian Telidon specification added a 150-bit/s frequency-shift keying (FSK) backward channel for the Bell 202 half-duplex recommendation.

Manufacturers' reluctance to build half-duplex capabilities into their low-cost terminals, however, has led to the mistaken belief in most of the United States that full-duplex communication is mandatory for remote terminal users. Although full-duplex communication is necessary for most computer-to-computer interaction that requires high bit rates in both directions, most remote terminal communications, including videotex, is characterized by high bit-rate inputs with low bit-rate outputs—an ideal situation for the half-duplex scenario.

Consider a typist who can reach speeds of 120 words per minute. Averaging 6 characters per word and 9 bits per character, the same typist could input data at 108 bit/s. Most half-duplex modem back channels run at a 75-bit/s minimum, and many are built to run at 150 or 300 bit/s—rates as fast as or faster than the fastest typist. Furthermore, a low-cost terminal with half-duplex communications capability is more than adequate for a person who wants to monitor home security, transact simple business, and entertain his family. On the other hand, a full-duplex modem is not cost-effective unless its capability to transmit in both directions is used.

Finally, a modem with associated hardware will remain more complex and more expensive than interface hardware and software that connect the modem at the videotex terminal CPU. Even as more modem types are integrated into VLSI components, the modem will still be the most expensive part of the terminal because the computational power required to implement it is far greater than that required for the half-duplex line turnaround at the interface.

Graphics standards

As mentioned above, several presentation-level standards exist worldwide. The oldest and most widely used is the British Prestel's. Prestel uses code sequences to represent characters and mosaic patterns, which create low-resolution graphic images. Figure 3 shows the four sets and the interaction of the codes defined in the Prestel protocol. The control codes have unalterable 7-bit representations. The second set of codes change the attributes of the display and are only valid following an ESC control code. The alterable display attributes are the color, height, or type of the characters. The selection of a character-type attribute alters Prestel's non-control representations to be either alphanumeric or mosaic codes. These character codes are the third and fourth sets of the Prestel protocol. The sets share common representations with the attribute codes depending on the preceding ESC or attribute character in the video memory. When the alphanumeric attribute is selected in the Prestel code, the alphabetic

The OSI of the ISO

The Open Systems Interconnection (OSI) model is used to describe network protocols that allow architecture-independent connection of computers for communication. As long as all machines in the network understand the protocol, it is possible to have direct communications between different computer types. The OSI model of the International Standard Organization (ISO) consists of seven protocol levels, each of which interacts with its next highest and lowest level. Thus, the level 7 interface is to the user, while the level 1 interface is the physical interface to the transmission media. The seven are:

■ Level 1, the Physical Layer, must assure that the bits transmitted over the physical media arrive with a minimum number of bit errors. The bit information is often modulated or demodulated into an analog carrier. Modems generate a voiceband analog carrier, whereas radio broadcast transmitters for data communication generate very high frequency carriers (in the MHz range).

■ Level 2, the Data Link Layer, breaks the data stream up into blocks. These blocks are used to minimize the number of errors that propagate from layer 1 to the rest of the network. By attaching error-recognition information (such as a checksum) to the block, the data link layer can acknowledge a properly received block or request retransmission of an improperly received block. The physical and data link layers of the ISO model are typically implemented in hardware.

■ Level 3, the Network Layer, is basically a traffic controller. It accepts messages and routes them to their proper destination. The network layer is also responsible for handling information floods by determining message priority.

■ Level 4, the Transport Layer, combines the blocks of data received from the network layer so they may be passed to the session layer. Blocks may arrive from, or be sent to, several nodes to assure optimal transmission speed. It is the transport layer's job to assure that these blocks are placed in order when they are received. It also may be very cost-effective to multiplex several users into one network node. The transport layer is often part of the computer operating system, and tends to be much more device-independent than layers 1 through 3.

■ Level 5, the Session Layer, interacts with the user to establish a network connection. Logging on and off of a timesharing computer is an example. Commands to copy, delete, or create files are handled by the session layer. Computer security is normally handled by software at this level of the OSI model. The session layer is implemented in most computer operating systems.

■ Level 6, the Presentation Layer, is responsible for the transformation of user data into a form understood by all nodes in the network. Data compression and code interchanging are the two most common forms of presentation protocols. Graphics protocols perform data compression to represent complex shapes as simple bit patterns. Most of the presentation level protocols described in this article use a series of codes to determine the information being sent.

■ Level 7, the Application Layer, is mainly concerned with the uniformity of user commands throughout the entire network and within the individual user's computer.

By developing a simple set of commands, acquired human skills become transportable from computer to computer. For example, by standardizing on PASCAL as a software development language, a programmer's capability to implement programs on different computers does not include a language learning curve at the beginning of each project.

and numeric characters have the same codes as an ASCII representation.

The French Post Telephone and Telegraph Ministry (PTT) developed a protocol known as Antiope in the late 1970s. Similar to Prestel, Antiope uses alphanumeric and mosaic characters to generate video images. In addition, it uses a powerful tool known as dynamically redefinable character sets (DRCS), which allows periodic changes to the character set displayed by the terminal in order to form higher resolution graphic images. Another standard, Telidon, was developed by the Canadian Department of Communications. This protocol uses alphanumeric characters and geometric shapes to draw images on a CRT. The geometric shapes are drawn through a series of end points, which are connected by commands known as picture description instructions (PDI). Typical PDIs are arc, rectangle, circle, square, and line.

In 1981, AT&T published a specification for an American videotex code called Presentation Level Protocol (PLP). PLP incorporated the alphanumeric and mosaic character sets, DRCs, and PDIs. The concept of a macro-PDI was introduced by PLP. A macro-PDI is a series of single-instruction PDIs which define an image that is to be drawn repeatedly. By issuing a single macro-PDI command, a complex video image can be generated. PLP was modified slightly and became the North American Presentation Level Protocol Syntax (NAPLPS). More about this protocol later.

Building it

The major functional components of a videotex terminal include the terminal function controller (CPU), the CRT controller, the modem, and the support circuitry required for each of these VLSI components (Fig. 4).

A terminal function controller, which can be a single-chip microcomputer or a microprocessor, is essential to a videotex terminal. The function controller coordinates the user inputs, the data for the images fed to the video display controller, and the information for the data communications component (modem). By assigning the various I/O ports available on a single-chip

3. Prestel. *An ESC from the control code set indicates that the next byte of data will define a particular attribute of the character—height, color, or background, for example. Then, depending on which character is selected, the terminal will select the alphanumeric or the mosaic character set.*

PRESTEL CODE FOR U.K. VIDEOTEX SYSTEM

LSB's (HEX) \ MSB's (HEX)	0	1
0	NUL	
1		CURSOR ON
2		
3		
4		CURSOR OFF
5	END	
6		
7		
8		
9		
A		
B		ESC
C	CLEAR SCREEN	
D		
E		
F		

CONTROL CODE SET (UNALTERABLE REPRESENTATIONS)

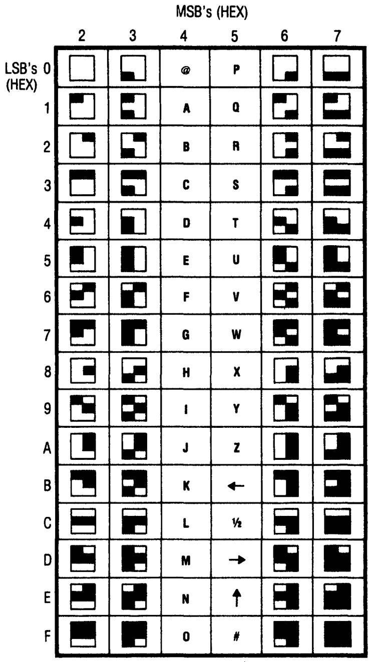
LSB's (HEX) \ MSB's (HEX)	2	3	4	5	6	7
0						
1			ALPHA RED	MOSAIC RED		
2			ALPHA GREEN	MOSAIC GREEN		
3			ALPHA YELLOW	MOSAIC YELLOW		
4			ALPHA BLUE	MOSAIC BLUE		
5			ALPHA MAGENTA	MOSAIC MAGENTA		
6			ALPHA CYAN	MOSAIC CYAN		
7			*ALPHA WHITE	MOSAIC WHITE		
8			FLASH	CONCEAL		
9			*STEADY	*CONTINUOUS MOSAIC		
A				SEPARATED MOSAIC		
B						
C			*NORMAL HEIGHT	*BLACK BACKGROUND		
D			DOUBLE HEIGHT	NEW BACKGROUND		
E				HOLD MOSAIC		
F				*RELEASE MOSAIC		

ATTRIBUTE CODE SET

*DEFAULT VALUES

LSB's (HEX) \ MSB's (HEX)	2	3	4	5	6	7
0	SPACE	0	@	P	-	p
1	!	1	A	Q	a	q
2	,,	2	B	R	b	r
3	t	3	C	S	c	s
4	$	4	D	T	d	t
5	%	5	E	U	e	u
6	&	6	F	V	f	v
7	'	7	G	W	g	w
8	(	8	H	X	h	x
9	)	9	I	Y	i	y
A	*	:	J	Z	j	z
B	+	;	K	←	k	¼
C	,	<	L	½	l	ǁ
D	-	=	M	→	m	¾
E	.	>	N	↑	n	—
F	/	?	O	#	o	■

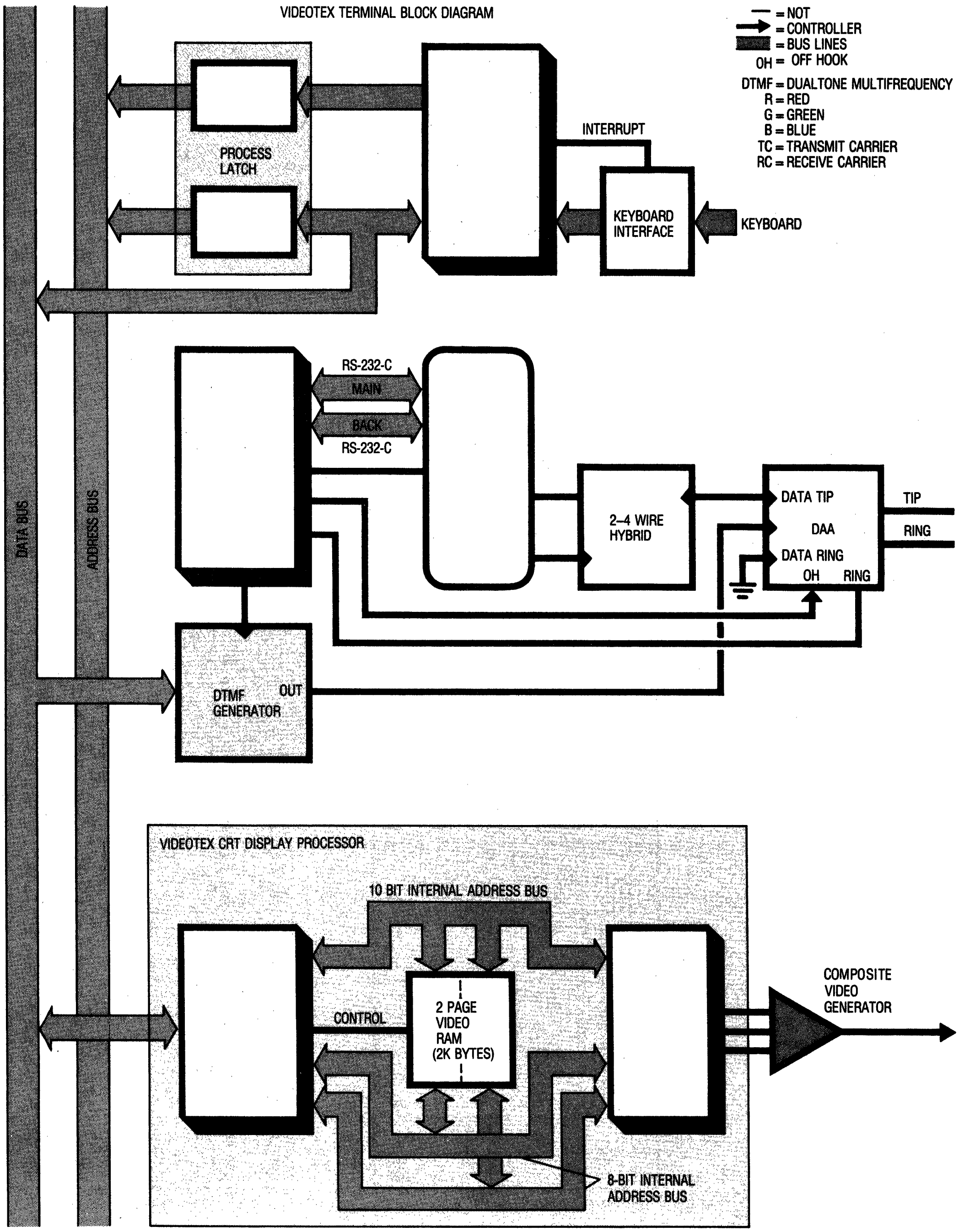
ALPHANUMERIC CHARACTER SET



LSB's (HEX) \ MSB's (HEX)	2	3	4	5	6	7
0		🬏	@	P	🬞	🬭
1	🬀	🬐	A	Q	🬟	🬮
2	🬁	🬑	B	R	🬠	🬯
3	🬂	🬒	C	S	🬡	🬰
4	🬃	🬓	D	T	🬢	🬱
5	🬄	▌	E	U	🬣	🬲
6	🬅	🬔	F	V	🬤	🬳
7	🬆	🬕	G	W	🬥	🬴
8	🬇	🬖	H	X	🬦	🬵
9	🬈	🬗	I	Y	🬧	🬶
A	🬉	🬘	J	Z	▐	🬷
B	🬊	🬙	K	←	🬨	🬸
C	🬋	🬚	L	½	🬩	🬹
D	🬌	🬛	M	→	🬪	🬺
E	🬍	🬜	N	↑	🬫	🬻
F	🬎	🬝	O	#	🬬	█

MOSAIC CHARACTER SET

4. Common bus. *The function controller, communication control unit, and the display processor all share a common data and address bus. Note that, since keyboard inputs can be either control or presentation data, that data is latched before it is placed on the bus so the CCU can decipher character sequences.*



microcomputer to each of these tasks, it is possible to provide all of the required functions in a single 40-pin package. For example, the Intel 8051 single-chip microcomputer (second-sourced by Advanced Micro Devices as the Am8051) has four 8-bit user-definable ports which can be used to control the three functional components of the videotex terminal. Three of the four I/O ports (P0, P2, and P3 in Figure 4) are used to support standard microprocessor interface functions.

The modem in this videotex terminal is the Advanced Micro Devices' Am7911, which implements all four Bell and CCITT FSK modulators in a single 28-pin package. The essential control signals in RS-232-C and CCITT V.24 serial interface are assigned to pins on the Am7911. In addition, the Am7911 allows simple selection of any Bell or CCITT modem type through a 5-bit mode control port. Modes for both 5-bit/s and 150-bit/s Bell 202 back-channel types are included on the Am7911, which also contains an improved CCITT V.23 back channel to allow transmission at both 75 bit/s and 150 bit/s.

The modem performs all analog-to-digital (ADC) and digital-to-analog (DAC) conversions on a single chip. The DAC circuitry has the ability to drive the 600-ohm telephone line impedance directly, thus eliminating the need for a drive buffer. All digital signal processing for transmit and receive signals is onboard the Am7911; and, because the modem is a digital process, it can hold the level of all frequencies generated on-chip to within plus or minus 0.5 db. This is particularly important when interfacing with European networks.

Communications functions, however, are not limited to data transmission. Features such as automatic calling, automatic redial, automatic disconnect, telephone number storage, and conversion from a parallel microprocessor data format to a serial modem format all should be included in the videotex terminal. The parallel-to-serial data conversion and control of the RS-232-C and CCITT V.24 signals on the modem are normally done by a component known as a universal asynchronous receiver/transmitter (UART). The UART functions, as well as automatic answering and progress tone (busy, dial tone) detection, could be incorporated into a single-chip microcomputer.

Middle manager

In Figure 4, this possible VLSI component is referred to as the communication control unit (CCU). Before communication is established, the 8051 function controller need only set the duration of the CCU tones and direct the CCU to establish a telephone connection. The CCU would be accessed only during normal operation to alter the tone duration settings, to disconnect the terminal from the telephone, or to transmit or receive data from the modem.

Specifically, the CCU would manage the functions of the single-chip modem and the data access arrangement (DAA) as well as the automatic connect and disconnect telephone functions. The CCU should be able to pulse and dual-tone multi-frequency (DTMF) dial in an auto-dialing mode. A programmable number of dial retries and a programmable interval between these retries should be supported by the CCU. Three types of disconnect could be provided: loss of carrier disconnect, loss of data (continuous mark) disconnect, and forced disconnect. The CCU would be connected to the 8051 terminal function controller via an 8-bit bidirectional data bus. Two UARTs (one main channel and one back channel) and a 5-bit port to control the Am7911 mode controls are built into this CCU, greatly simplifying the interface to the Am7911.

The two- to four-wire hybrid, or duplexer, connects the four-wire modem—receive carrier (RC) and ground, and transmit carrier (TC) and ground—through the two-wire DAA to the telephone network. The hybrid must match the network with its own balancing impedance (between TC and RC). Since network impedance varies from connection to connection, a good model of the network variance must be developed, and the hybrid balancing network must closely match this model.

The DAA protects user and network equipment from each other when one or both ends are not functioning properly. The DAA restricts the transmission output of the user equipment and isolates one end from the other via a transformer. Large voltage transients, such as those from lightning, must be diverted by the DAA to protect the modem.

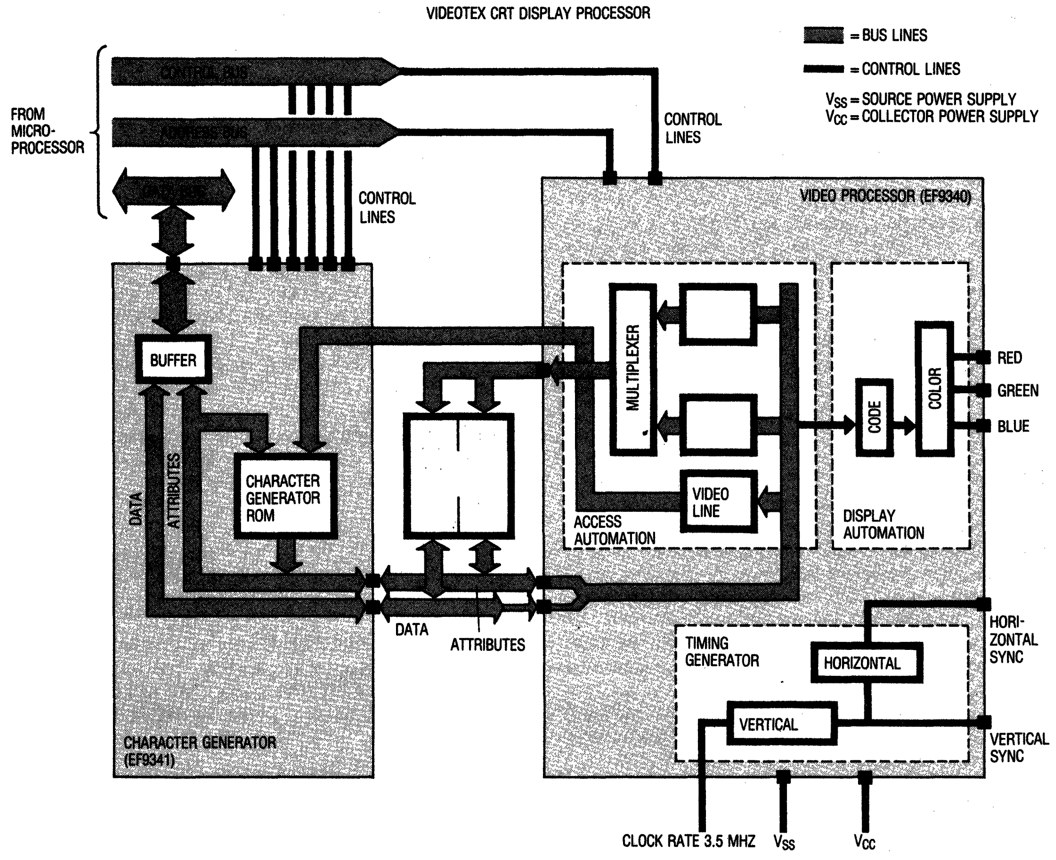
Finally, the DAA must provide on-hook/off-hook impedance and current levels to ensure that the network equipment functions properly. Each DAA design is registered with a national body governing the telephone network. In the United States this body is the Federal Communications Commission whereas each European country has its own PTT ministry. Before any DAA design work is undertaken, it is advisable to contact one of these national governing agencies.

Display processor

Many of the CRT controller chips available are quite sophisticated and require a minimum of CPU overhead. An example is the CRT controller chip set for the terminal in Figure 4. The Thomson-Efcis EF9340/41 is a two-chip CRT controller, which provides a maximum of 25 rows by 40 columns of alphanumeric and mosaic characters on a raster scan CRT. This CRT controller supports DRCS with the addition of external RAM. To set up the EF9340/41 before any data has been displayed on the CRT, the 8051 function controller first initializes the CRT controller's registers. After initialization, the 8051 is responsible only for the periodic update of these registers and the flow of data into the video RAM. For the alphanumeric and mosaic display, each character on the CRT screen maps into a specific location in the video RAM.

The Thomson-Efcis EF9340/41 CRT controller chips are two 40-pin NMOS VLSI circuits (Fig. 5). The EF9340 contains a timing generator, a display automaton, and an access automaton. The timing generator controls which of the two automata are running at any given time. Figure 5 illustrates this timing sequence between the access and display automata. On a raster scan CRT, the characters displayed do not occupy the full trace time of the electron gun. If the display were a

5. Automaton. *The video processor has basically three functional units—the access automaton, the display automaton, and the timing generator. During the overscan and retrace periods, the access automaton is operational; during regular display periods, the display automaton is operational.*



window, it would extend for several characters above, below, and on either side of the visible display because of difficulties controlling the exact points where the electron gun completes its tracing across the display. The periods of time when CRT's electron gun is outside of the visible display are known as overscan times.

Two other factors contribute to the border around the visible display: the time it takes for the gun to move from the right to left edge at the beginning of a new line, and the time it takes for the gun to move from the lower right to the upper left corner of the CRT at the start of a new display frame. These are called retrace times. The EF9340 access automaton operates during the overscan and the retrace periods, whereas the EF9340 display automaton operates during regular display periods.

Character and attribute codes are read from the video memory. Each character code corresponds to a series of dot patterns in memory. The EF9341 contains 256 sets of dot patterns in a character generator ROM. Some of the character codes are reserved for use with external character generation memory; this feature allows the terminal designer to add DRCS features.

Each standard character code addresses a byte of memory in the EF9341 character generator ROM. The seven-bit attribute determines the foreground and background color of the character; whether the character is double height, width, or both; whether the character is reverse video; and whether the character is blinking. From the character generator output and the attributes, red, green, and blue video signals are created as outputs from the EF9340.

Two additional outputs are generated by the EF9340 timing generator: horizontal sync and vertical sync. Horizontal sync is a signal at the end of each display line that forces the CRT to retrace to the next line. Vertical sync is generated at the end of a display frame to start the CRT trace once again. Generation of a composite color video signal is accomplished by adding two carriers to the horizontal and vertical sync signals. The first carrier, called the luminance, is associated with the brightness of a particular dot. The luminance is an amplitude-modulated signal, which varies from nearly black to nearly white. A monochrome display would use the luminance carrier exclusively as its input.

The second carrier is the chrominance, which determines the color of a particular dot. Chrominance is a 3.579545 MHz carrier that is phase-shifted from a reference carrier to determine which of the three guns (red, green, blue) of the CRT is fired. By combining the two carriers with the two amplitude-modulated sync signals, a composite video signal suitable for direct input to a color monitor can be created.

ASCII terminals

A different class of videotex network is the provider of strictly textual information such as The Source (TM) and Compuserve (TM) in the United States. These two networks currently support Bell 103 modems, but not AT&T 202 modems, because the low-cost ASCII terminals that most users buy have only one baud rate for transmission and reception. The half-duplex backward channels cannot be used in this situation unless an intelligent communications network is built.

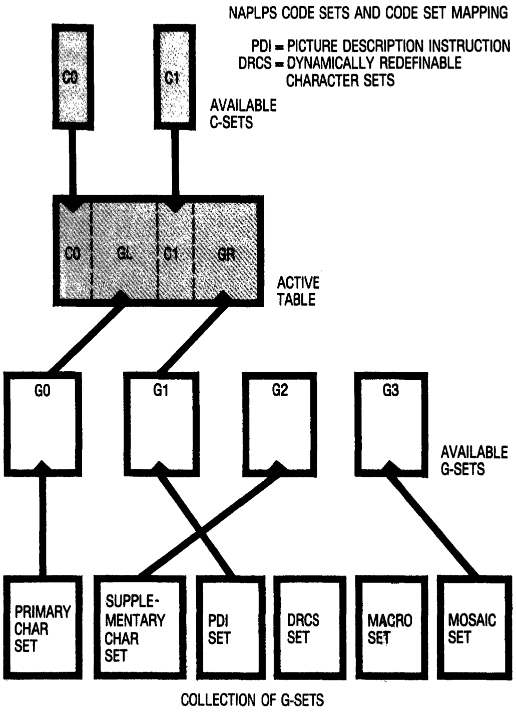
A strictly ASCII terminal could be implemented similarly to the graphics terminal referred to in Figure 4. However, since all that is involved is displaying alphanumeric characters, display functions could be handled using one of the 8051's 8-bit user-definable I/O ports which connects to an RS-232 driver and receiver that interfaces to the video display.

More complex functions continue to be integrated into VLSI components. CRT controllers that can generate high-resolution, high-speed color graphics are certain to appear soon. As several new graphics standards gain support, integrated circuit manufacturers will combine presentation level protocol handlers with high-performance CRT control to build fully integrated video display controllers. These components will be highly applicable to videotex terminals and color graphics terminals alike.

Once a single presentation level protocol gains unified support, a VLSI protocol controller is far more likely to appear. Toward that end the NAPLPS graphics protocol is gaining tremendous support. Most of the principles developed from previous protocols have been incorporated into NAPLPS, yielding a powerful set of graphics control statements. This protocol uses a code-swapping method quite similar to that of the Prestel alphanumeric and mosaic codes.

NAPLPS is an 8-bit code (as opposed to 7-bit for Prestel), which defines two control sets (C-sets) consisting of 32 codes and two graphic/textual sets (G-sets) consisting of 96 codes. Figure 6 illustrates the interaction between the different code sets in NAPLPS. The two active G-sets are selected from four different available code sets. The four available G-sets are swapped from the entire collection of G-sets. One reason for this layered approach to code swapping is

6. NAPLPS. *G-sets are graphic/textual code sets. C sets are control codes. At any time the two active G-sets are selected from the four available sets.*



that the highly used, available code sets can be stored in memory constantly, while the codes used less often are kept in slower, less expensive secondary memory (possibly on a disk). This reduces the cost of the memory required to implement a complete NAPLPS character set, while still offering a choice of four quickly interchangeable code sets.

Although a mechanism exists for swapping C-sets, there are only two available. These occupy the C-set locations in the NAPLPS protocol. The standard NAPLPS alphanumeric codes have almost exactly the same representations as in ASCII. This code set is known as the primary character set in NAPLPS, and is one of the active default G-sets in NAPLPS. The second active default G-set is the PDI. The two other available default code sets are the supplementary character set and the mosaic set. Other G-sets in the collection include a DRCS set and a macro set. An integrated NAPLPS controller will probably be available some time in the future. ■

Stevan Eidson, an engineer in product planning and applications for AMD's Data Communications Department, received an MSEE from UCLA.

Jeffry L. Parker, Advanced Micro Devices,
Sunnyvale, Calif.

Third-generation codecs pave way for future digital networks

The use of high-density, economical integrated circuitry has made possible new, accurate digital signal processing.

By the year 2000, all-digital telephone networks are expected to be commonplace throughout the United States. In the meantime, a critical factor in such an evolution will be the development of flexible, economical network components that can be easily programmed to match the electrical requirements of various applications and evolving network standards.

Advances in coder-decoder technology, better known as codec filters, will be key to reducing the cost of the elements that encompass the functions of line interface, digital switch, and controller. In fact, due to increasing integration of on-chip line-interface functions alone, it is expected that the cost per digital line will decrease by 15 percent annually over the next three to five years.

A digital-switching network routes calls between telephones using a binary representation of each voice signal. The line interface, which is connected to the telephone through two wires, supplies approximately 48 volts direct current to power the telephone. It provides a 20-hertz (Hz), 100-volt alternating current signal to ring the telephone and carries the voice signals to and from the telephone. Transformer hybrids or subscriber-line interface circuits (SLICs) separate the bidirectional voice signals on the two-wire line into a four-wire transmit-and-receive signal.

The codec then samples, at 8 kHz, the analog transmit telephone signal and assigns a digital pulse-code modulated (PCM) word, usually 8 bits long, for each sample. It also reassembles the PCM receive data into an analog signal [see "ADPCM standard"].

The PCM codes are usually carried to the remote switch via a digital T1 link. In the United States, 24 voice channels are carried on each link at a data rate of 1.544 Mbit/s. The data rate is derived as follows: 8 bits (PCM code) X 24 (conversations) + 1 (framing bit) X 8 kilohertz (kHz) (sampling rate). Each 193-bit block of voice channels is called a frame, and the position of a channel within a frame is called the time slot. Using a T1 multiplexer, the transmitted PCM data from the codec is put into the correct time slot for that channel's transmit data; a channel is switched between two phones by taking the transmitted PCM data from one digitized telephone line and placing it in the other's receive time slot on a T1 link. Conversely, receive data is taken from the appropriate time slot on a reverse-direction T1 link.

In the codec-sampling circuitry, low-pass transmit filters prevent out-of-band frequencies (frequencies greater than half the sampling frequency, or 4 kHz) from inadvertently being sampled and included as the transmitted, digitized voice signal. This type of sampling error is called aliasing. Similarly, the unwanted high frequencies in the decoded receive signal resulting from the decoder's reconstruction process can be removed by low-pass receive filters.

At the two- to four-wire interface, the transmit signal frequently gets reflected into the sender's receive path, and talkers hear this as an echo of their own voice. The echo becomes more objectionable as the round-trip delay between actual speech and the electronic echo increases.

Traditionally, a device called a balance network was used with the subscriber-line interface circuit hybrid to attenuate the echo. This component, which consists of resistors and capacitors, is a complex impedance that attempts to match the telephone-line impedance. Third-generation devices accomplish this by using digital filters that process the signal after it has been digitized (Fig. 1). If there is a perfect match, the echo is eliminated. In practice, however, the match is often

ADPCM standard

The International Telegraph and Telephone Consultative Committee (CCITT) recently standardized a transcoder for converting a 64-kbit/s pulse-code modulation (PCM) channel to or from a 32-kbit/s channel. The conversion is applied to the PCM bit stream using an adaptive differential pulse-code modulation technique, or ADPCM. The intention of the standard is to introduce 32-kbit/s ADPCM into the public-switched telephone network and into places where 64 kbit/s is currently used for both voice and voice-band data.

The ADPCM algorithm is the result of a joint effort of experts from the United States, Canada, France, Great Britain, and Japan. The goal was to derive a uniform 32-kbit/s standard throughout the world—in contrast to the different A-law and mu-law companding standards now used with 64-kbit/s PCM. The algorithm was extensively tested for both voice and nonvoice signals; subjective voice tests were conducted in the United States, Britain, France, Germany, Japan, Italy, and China. The ADPCM transcoder has passed benchmark tests on various modems. Further, the CCITT is continuing efforts to standardize transcoding equipment like echo cancelers and to assess their impact on evolving integrated services digital network standards (ISDN). Also under study is a performance specification for 32-kbit/s codecs. In time, 32-kbit/s codec chips will replace the existing 64-kbit/s PCM codecs.

poor with first- and second-generation devices. The average echo attenuation is only 6 to 12 decibels (dB), when 20 dB or more would be desirable.

Also with the earlier devices on long transmission paths, such as satellite links, the echo is so noticeable that echo suppressors are required. These devices suppress echo by detecting when one party is talking and completely cutting off the opposite voice path until the conversation is reversed.

The termination impedance of the local telephone-line interface must match the actual line impedance in order to prevent echoes on the two-wire side. This termination impedance is traditionally set by using a transformer hybrid with the correct impedance or a subscriber-line interface circuit. The impedance values, of course, will vary depending upon the characteristics of the telephone line.

To prevent instability and singing (oscillation) in the four-wire lines, it is necessary to adjust the gain through the telephone network. This is accomplished by providing gain in the transmit path and loss in the receive path. Gain is provided to maintain signal strength over long distance. However, if this gain were left in at the far end, it would loop around and cause oscillation on the receive path. Consequently, resistors on the receive path need to be adjusted to generate loss. Traditionally, gain and loss functions on the four-wire circuit have been accomplished using a resistor voltage divider to generate loss and an operational amplifier (op-amp) to create gain.

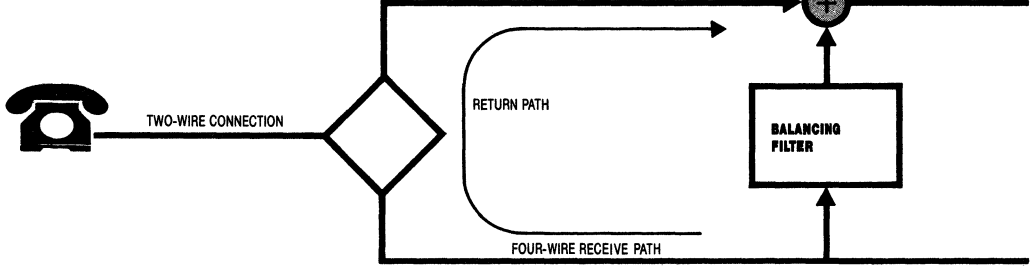
Codec standards

To allow customer equipment to connect to the telephone network and to facilitate telephone communications between countries, standards have been established for the transmission characteristics of telephone equipment. The major standard-setting body is the International Telegraph and Telephone Consultative Committee (CCITT). However, each country's telephone administration may establish standards to meet their individual needs.

The CCITT is one of the consultative committees of the International Telecommunication Union (ITU). Member nations of the ITU, numbering over 150, make recommendations to the committees as well as vote on proposed standards. A large number of major producers of communications equipment also contribute to the committees as nonvoting members.

In order for a codec to be used in equipment that connects with common carriers, it typically must meet both AT&T and CCITT specifications. These standards apply to the complete switching network measured

1. A split. *The hybrid splits the two-wire signal into separate four-wire transmit and receive signals. However, some of the receive signal leaks into the transmit path through the return path. Third-generation devices like the Am7901A balancing filter removes any receive signal that has leaked into the transmit path.*



between two subscribers. The measurement path includes two codecs (one per subscriber at each end) and the remaining line interface circuitry. To allow for some degradation of the voice signal by the line interface, the transmission characteristics of the codec must be better than twice as good as the overall network requirements.

The ideal codec would not modify the transmitted signal in any way, and the standards bodies try to approach this ideal. Some of the more important measurements of transmitted signal distortion include:

- Attenuation distortion,
- Variation of gain with input level,
- Total distortion,
- Idle-channel noise, and
- Out-of-band signal specifications.

Attenuation distortion. This is a measurement of the variation of the codec's gain using input signals of varying frequency. Ideally, the gain from analog-input to PCM-output (and vice versa) should be 0 decibels, or no change in amplitude. However, the codec filters used are not perfect, and the gain will vary depending on the frequency.

Figure 2 shows the CCITT specification for attenuation distortion. The accepted voice bandwidth for telephone transmission is 300 to 3,400 Hz. In the middle of this range, the variation of gain must be rather small, but near the band edges (2,400 to 3,000 Hz, for example) the attenuation or loss can increase. The specification does not consider the absolute gain through the codec. It is only concerned with how the gain varies relative to a reference point. The measurement is made by comparing the absolute gain at a midband frequency with the absolute gain at other frequencies.

Variation of gain with input level. Also known as gain tracking, this parameter measures gain variation of the codec for different input signal levels at the same frequency. This is a good measure of how well the codec can reproduce signals at low levels. Figures 3A and 3B show the CCITT specification limits for gain variation. The measurement is made using a reference level of −10 dB. Note that the limits become less stringent at lower input signal levels.

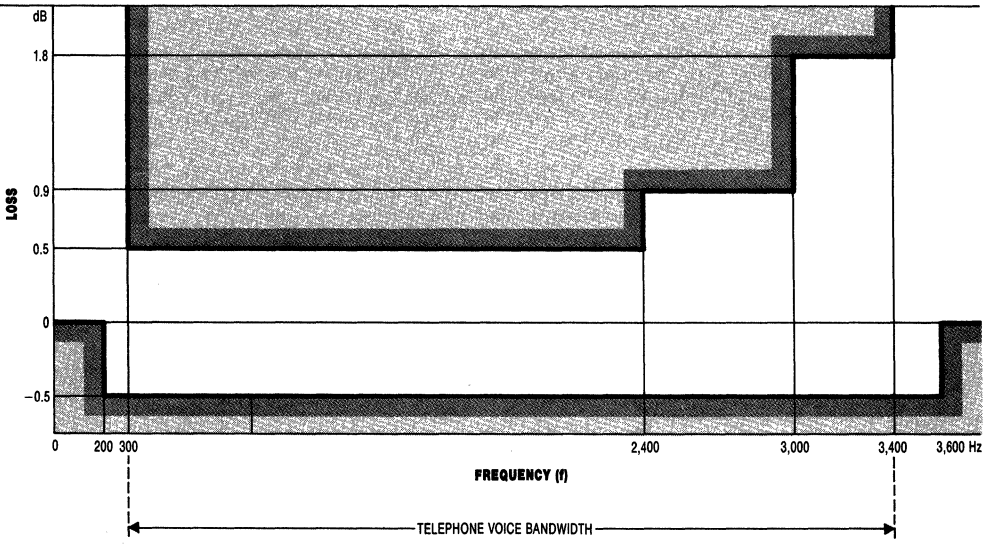
Two methods are used to measure gain tracking. Method 1 (Fig. 3A) uses a predetermined noise signal for the input. The contents of the noise signal are clearly defined by the CCITT and are meant to closely simulate the energy content of a typical voice signal. Method 2 (Fig. 3B) simply uses a single frequency tone as the input signal.

Total distortion. As a measure of the codec's signal-to-noise characteristics, total distortion includes the distortion that is caused by the 8-bit PCM encoding compression [DATA COMMUNICATIONS, June 1984, p. 133]. The measurement indicates the noise level generated in the codec relative to the desired signal.

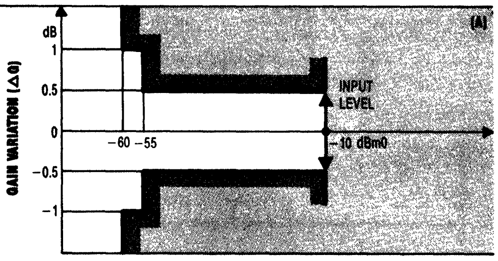
When talking over a telephone, total distortion is sometimes heard as a hiss. The lower the signal-to-noise ratio, the more difficult is it to understand the speaker at the other end. The 8-bit PCM compression causes a distortion that limits the maximum signal-to-noise ratio. As with gain tracking, one measurement method uses a specific noise signal, and the other method uses a single frequency tone.

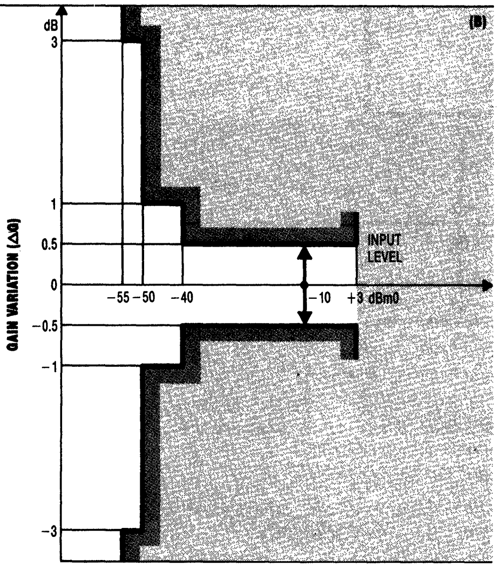
continued

2. Tight specifications. *CCITT specifications for attenuation distortion allow little variation in gain (plus or minus 0.5 decibels) in the middle of the voice bandwidth (300 to 2,400 hertz) but near the edges (2,400 to 3,000 hertz) the CCITT attenuation specification allows variances to increase.*



3. Tracking at lower levels. *At lower input levels, the CCITT specification for "variation of gain with input level" is considerably relaxed.*





Idle-channel noise. This measurement specifies how much noise can appear at the codec's output when there is no input signal. The noise is generated within the codec from such sources as internal clocks and auto-zero circuits. A telephone user will hear this as a background hiss when there is no conversation.

The measurement is made with the transmitter connected to the receiver; for the transmitter, noise measured must be at least 65 dB below the level of a 0 dBm0 (decibel milliwatts measured from a zero transmission level point, or TLP—usually that of the toll switchboard) signal. For the receiver, the noise must be at least 75 dB below a 0 dBm0 signal. Before making the noise calculation, the idle-channel output signal is weighted so that frequencies outside of the voice bandwidth are ignored.

Out-of-band signal specifications. These limits are concerned with signals outside of the voice bandwidth. The test, which is called discrimination against out-of-band input signals, requires that an analog signal beyond the range of 100 Hz to 3,400 Hz not cause a digitized signal to be induced within the voice band unless that signal is below a certain level. Conversely, the test called the spurious out-of-band signals at the analog output sets a limit on the level of any resultant digitized signal that is outside of the voice bandwidth but was caused by an analog input signal within the voice band.

Standard rate and coding

The CCITT has also specified the sampling rate and the encoding schemes to use for the digital PCM data. The sampling rate was chosen as 8,000 samples per second, so that signals in the voice-channel band of 300 Hz to 3,400 Hz can be accurately reproduced. This is so since, according to the Nyquist sampling theorem, a band-limited signal sampled at a rate equal to, or higher than, twice the highest significant frequency component will yield a resultant sample containing all the information of the original signal. The 8 kHz sampling rate can thus reproduce signals up to 4 kHz. But to simplify the band-limiting filters, the voice band only extends to 3,400 Hz.

Additionally, two 8-bit encoding/companding schemes, A-law and mu-law, have been standardized by the CCITT. Mu-law is used in North America and Japan, and A-law is used by the rest of the world. The encoding, or compression (A-law or mu-law), allows an 8-bit byte to represent a speech signal with a wide dynamic range (72 dB) that would normally require 13 bits to represent. This reduces the bandwidth required for the digital data. Speech contains more low-amplitude than high-amplitude signals. Thus, the encoding schemes use a logarithmic transfer function that has more steps at low signal levels and fewer steps at high levels. This allows for a better signal-to-noise ratio at low levels, at the expense of a poorer ratio at the less-frequent high levels.

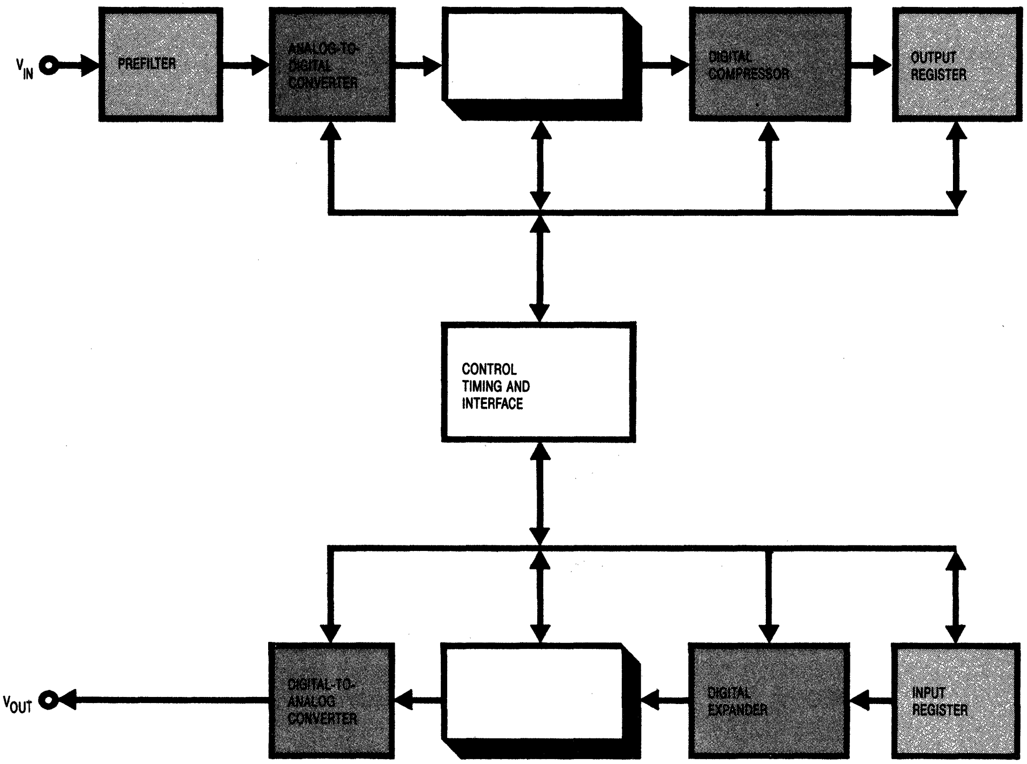
Previous codec and filter solutions

First- and second-generation codec and filter integrated circuits (ICs) have essentially been analog solutions for the digital-switching SLIC. First-generation products use two ICs for the filter and codec. Second-generation products combine the filter and codec into one IC. A switched-capacitor filter technique implements the transmit and receive filter sections.

In first- and second-generation codecs, the analog-to-digital (A/D) and digital-to-analog conversions (DACs) are done by the codec section. Since the sampling rate is only 8 kilohertz, the clock cycle is split to share one digital-to-analog converter (DAC) between the transmit and receive paths. The DAC function includes the companding of the 8-bit PCM code word. The codec transmit section uses a sample-and-hold and successive-approximation register in conjunction with the DAC to perform the A/D function during a portion of the 8 kHz sampling period.

continued

__4. Filtering signal.__ Two independent digital signal processors perform the required filtering in the Am7901A SLAC. Early high-speed A/D conversion is followed by digital filtering that reduces the sampling rate to 8 kHz. By filtering the signal after it has been digitized, the noise generated by the auto-zero circuitry is minimized.



A capacitor external to the codec is required for the sample-and-hold circuit. The receive side also has a sample-and-hold circuit that is used with the DAC to generate the analog output signal. An auto-zero circuit removes any offset voltage in the input transmit signal and an external capacitor stores the offset voltage (an offset voltage is the difference in voltage at the inputs of an operational amplifier).

In first-generation codecs, gain adjustment and two-to-four wire transformer hybrid drive are provided in the filter integrating circuits. The differential inputs and the output of the transmit filter op-amp are brought out to pins on the chip. This allows external resistors to be used to set the gain on the transmit signal. Conversely, loss is introduced into the receive signal by using a resistive voltage divider. In order to drive low-impedance loads, a dedicated op-amp is provided on chip. This can be used to feed a 600-ohm transformer hybrid directly.

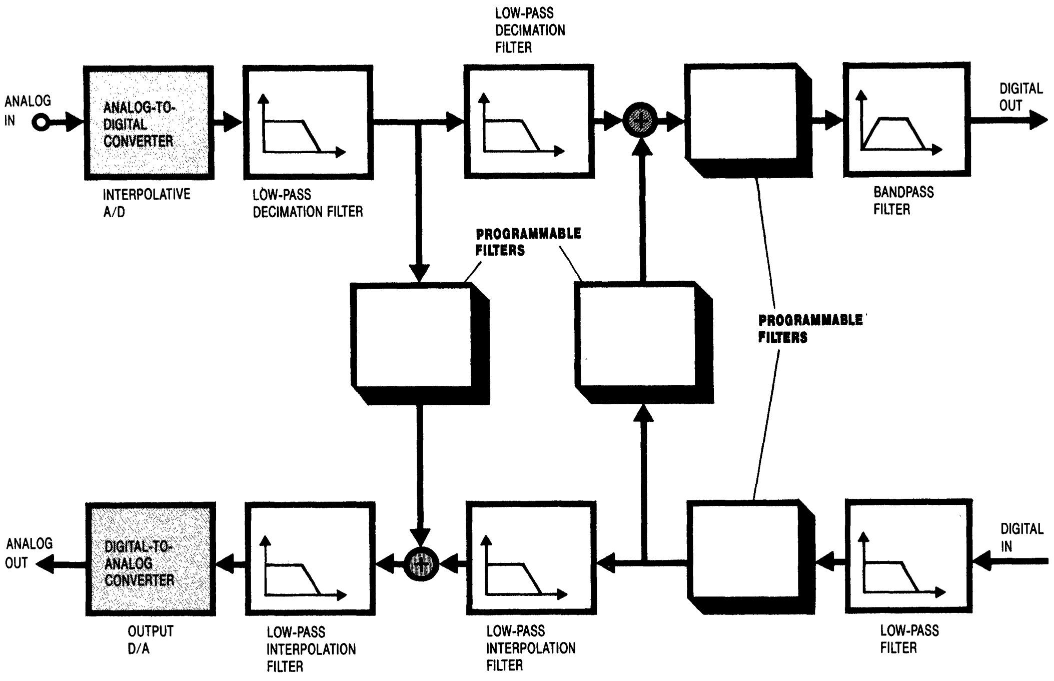
In first- and second-generation codecs, the time-slot assignment circuit has generally been a separate IC that interfaces between the codec and the PCM highway or T1 link. It takes the 8-bit PCM code from the transmit section and puts it into the correct time slot of the transmit PCM highway. The time slot performs a similar function for the receive side by taking the PCM data from the selected time slot of the receive PCM highway and feeding it to the codec. The correct time slots are programmed via an external controller.

Third-generation solutions

New devices such as the Am7901A Subscriber Line Audioprocessing Circuit (SLAC) eliminate the problems associated with analog filters by using digital signal processing techniques. This permits the efficient processing of analog data by first converting the signal to a digital format, then filtering it digitally, and finally converting it back to an analog signal. The use of high-density, cost-effective digital very large-scale integration circuitry has made this process economically feasible.

The transmit and receive sections of the Am7901A are implemented as two independent signal processors (Fig. 4). For the transmitter, an early, high-speed A/D conversion at 512 kHz is followed by digital filtering that reduces the sampling rate to 8 kHz. In addition, it

5. Programmable filters. *Digital signal processors allow implementation of programmable filters and gain adjustments. Rather than using a balancing network, the trans-hybrid balancing filter can be programmed for echo attenuation. A filter modifies the termination impedance of a SLIC or transformer hybrid.*


ANALOG IN
ANALOG-TO-DIGITAL CONVERTER
INTERPOLATIVE A/D
LOW-PASS DECIMATION FILTER
LOW-PASS DECIMATION FILTER
PROGRAMMABLE FILTERS
PROGRAMMABLE FILTERS
BANDPASS FILTER
DIGITAL OUT
ANALOG OUT
DIGITAL-TO-ANALOG CONVERTER
OUTPUT D/A
LOW-PASS INTERPOLATION FILTER
LOW-PASS INTERPOLATION FILTER
LOW-PASS FILTER
DIGITAL IN

provides the proper band-pass filtering.

By filtering after the signal is digitized, the noise generated by the auto-zero circuitry is minimized. This significantly reduces idle-channel noise. The receive side uses digital filters to filter the low-pass PCM input signal and increase the sampling rate before the high-speed D/A conversion. Time-slot assignment logic is provided to interface to the PCM highways. This architecture limits the analog circuitry to less than 10 percent of the IC chip area.

Advantages

The digital signal processors (Fig. 5) enable the easy implementation of several user-programmable filters and gain adjustments. Rather than using a balancing network, the trans-hybrid balancing filter can be programmed for echo attenuation. An impedance-matching filter modifies the effective termination impedance of a SLIC or transformer hybrid.

The impedance-matching filter can cause attenuation distortion in the receive path, which is corrected by the frequency adjust filter. The gain in either path can be adjusted using a software command for increments that are smaller than 0.1 dB, providing 0-to-12 dB gain for the transmit side and 0-to-12 dB loss for the receive side. It is because the increment codes are digital that the gain is set exactly.

Digital signal processing (DSP) allows for the relatively easy and quick design of new telecommunications products. The building blocks for these products (processor, control logic, A/D, and DAC) remain much the same as with analog devices. However, the algorithms that implement the functions of the digital building blocks can be simulated on a high-level computer in order to ensure a high degree of certainty that the actual chips will work the first time.

These same digital techniques can be applied to the upcoming ISDN (Integrated Services Digital Network) family of products. ISDN allows both voice and data to be transmitted over existing telephone lines. Voice is digitized at the subscriber terminal by devices such as the Am7930 Digital Subscriber Controller and is transmitted to the PBX or Central Office over digital subscriber lines. The audio processing functions (codec, filters, programmable gain, and tone generation) can be efficiently implemented using DSP techniques similar to those used for the Am7901A. ■

Jeffry Parker, a section manager at Advanced Micro Devices, holds a master of science degree in electrical engineering from Stanford University and a bachelor's degree from Lehigh University.

Design Of A Single-Chip Bell 212A Compatible Modem Employing Digital Signal Processing

David Taylor, Stevan Eidson
Advanced Micro Devices
901 Thompson Place
Sunnyvale, CA 94088

Abstract

Modems operating at 1200/300 bps full-duplex have to this point been implemented using a combination of microprocessors, analog hybrid filters, and analog demodulators. The goal of this project is to design a Bell 212A compatible modem with a Bell 103 fallback option in a single-chip CMOS VLSI component. Digital signal processing is employed for all major filtering, modulation, and demodulation tasks required of the modem. All of the DSP algorithms were designed to perform optimally with a shift-and-add type of architecture. This paper presents the algorithmic and system options that have been explored through the process of arriving upon the final design.

Introduction

The second generation single-chip modem is a 1200-bps full-duplex modem compatible with the Bell 212A specification. The modulation technique employed by Bell 212A compatible modems is known as differential-quarternary phase-shift keying (DQPSK). Included within the 212A specification is communication compatibility with the existing base of Bell 103 series modems operating at 300 bps. The modem on the answering end of a data call automatically senses the transmission speed (300 or 1200 bps) of the originating modem and adapts to that speed.

Key goals established for the 212A modem project were: all signal processing must be performed on chip, analog-to-digital and digital-to-analog conversion must be contained on chip, no external components should be required, and performance should be acceptable to customers who traditionally have relied on box-level 212A solutions.

In order to meet these goals, extensive thought was given to the telephone network environment, silicon technology to be used, architecture, signal processing compatibility of both the DQPSK and FSK modems, and algorithm development for efficient implementation.

Digital Signal Processing In VLSI Modems

All modem features (modulation, demodulation, filtering, interface control, etc.) must be integrated onto a single chip, so care must be taken in the design of the modem so as not to waste silicon. There are unique constraints placed on developing

VLSI silicon solutions relative to discrete implementations. The amount of analog circuitry on the chip must be limited because it is more sensitive to process variations, power supply, and noise considerations than digital circuitry.

Because of the feature set to be implemented, a digital-signal-processing architecture was chosen instead of using a switched-capacitor technology. A DSP implementation provides some significant advantages over switched capacitor technologies, especially for the 212A compatible modem. For instance, additional stages of filtering require relatively little additional silicon area (ROM and RAM only) in a DSP architecture. However, any additional stages of filtering in a switched-capacitor technology require modifications to the major portions of of the component layout. Taking this analogy further, once the ALU overhead in a DSP architecture is paid for, any additional processing costs very little. Future modifications and improvements are also easy to implement as they normally require simply changing the instruction microcode or filter coefficients stored in a ROM.

Most importantly, implementation of both the DQPSK and FSK modulation techniques required by the Bell 212A specification are easily implemented within a DSP architecture. Virtually identical microcode may be used for both modem types when thought is given to the algorithm development. The FSK algorithms may be designed as simply a subset of the DQPSK algorithms. The finite impulse response (FIR) and second-order infinite impulse response (IIR) filter structures are the same for both modem types, so coefficients from a coefficient ROM are the only major differences between the FSK and DQPSK modems.

The 212A-compatible modem using digital signal processing is a multiply-intensive processor. One multiplicand is often a fixed filter coefficient, while the other operand is the input data sample. Only a few times within the signal processing chain are multiplies required between data values. Since building a multiplier on chip is an expensive silicon investment, fixed-coefficient multiplies may be implemented within the ALU in a simple manner. Using a shift-and-add representation for these filter coefficients, the multiplication can be performed in two to four microcode instruction cycles. The techniques used for coefficient representation are described further on in the paper.

DQPSK and FSK Modulation

The two modulation techniques used within the 212A compatible modem (DQPSK and FSK) are incompatible with one another. FSK modulation is a digital form of frequency modulation where the analog carrier generated is a function of the baseband input signal. If the input is a binary "one", a particular frequency is generated. If the input is a binary "zero", a second frequency is generated. Figure 1(a) shows a binary sequence which will be used to modulate both an FSK and a DQPSK waveform. Figure 1(b) shows the time-domain FSK waveform as a function of a digital input sequence. The frequency spectrum generated by a random FSK modulation is an infinite spectrum. (Refs. 1,2)

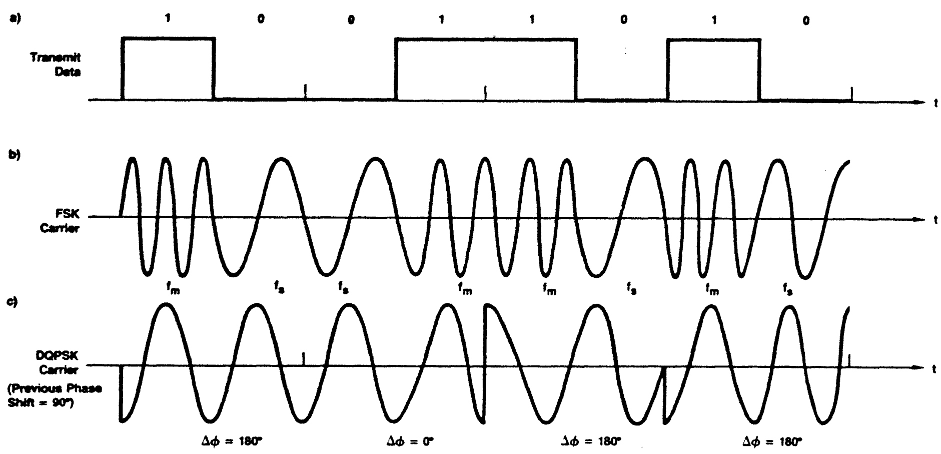

a)
1 0 0 1 1 0 1 0
Transmit Data
t
b)
FSK Carrier
f_m f_s f_s f_m f_m f_s f_m f_s
c)
DQPSK Carrier (Previous Phase Shift = 90°)
Δφ = 180°
Δφ = 0°
Δφ = 180°
Δφ = 180°

Figure 1: Modulating the digital data pattern in (a) is performed in two manners for the Bell 212A modem. (b) shows an FSK modulated waveform where f_s is the space frequency and f_m is the mark frequency. (c) shows an unfiltered DQPSK modulated waveform.

DQPSK modulation differs from FSK modulation in that DQPSK is a modulation of the phase of a constant carrier frequency. DQPSK modulation encodes two bits per modulation period (baud interval); hence, one of four phases transmitted during a modulation period represent the two bits encoded. Additionally, the phase shift performed on the carrier is differential in nature. In other words, the phase shift performed during the current baud interval is relative to the phase which was transmitted during the previous baud interval. Figure 1(c) shows a time-domain waveform which represents the unfiltered DQPSK signal. The time-domain expression for the DQPSK carrier in the current baud interval may be expressed as:

$$\sin(w_c + \text{phi} + n*\text{pi}/2), \qquad (1)$$

where w_c is the radian frequency of the carrier,
phi is the phase of the carrier transmitted during the previous baud interval
and n*pi/2 represents the phase shift performed during the current baud interval, where n = Ø,1,2,3.

Similar in nature to the randomly modulated FSK signal, the randomly modulated DQPSK signal also contains an infinite spectrum. (Ref. 3)

Normally, additional shaping is provided for the transmitted DQPSK signal. This shaping is provided to help limit the amount of inter-symbol interference (ISI) in the transmitted signal. The shaping normally obeys a cosine rolloff function which decays the tails of the baseband pulse to zero more rapidly than would normally occur with arbitrary band-limiting in the band-pass filter. The Bell 212A specification requires a 22% rolloff characteristic. The shaping may be implemented either in the baseband of the pulse or in the passband of the modulated signal.

Shaping is not provided for the transmitted FSK signal because of the asynchronous nature of the data. For pulse shaping to be effective, the data sequence must be synchronous.

The Public Switched Telephone Network (PSTN)

The telephone network is a harsh environment for modem operation. Problems such as lightning, induction from power line cables, switching noise caused by central offices, FDM and TDM conversion inaccuracies, plus a myriad of other noise-inducing problems may tax the ability of the modem to provide accurate data demodulation. The human ear tends to act as an effective low-pass filter and interpolator to ignore impairments of this type. However, if a dropout occurs during data transmission, the modem is likely to begin making decision errors. Hence, the design of a data modem such as the 212A must take into account possible harsh operating conditions such as those cited.

Typically, modem performance if characterized as a plot of bit error rate versus signal-to-noise ratio. Bit error rate is simply a measure of the number of errors received in a bit stream versus the total number of bits sent. Signal-to-noise ratio is a measure of the in-band signal power strength versus the in-band noise power strength. Figure 2 shows plots of the theoretical bit error rate performances of both the FSK and DQPSK modems. (Refs. 2,4) The assumption is that the noise is additive gaussian white noise (AGWN). While in practice it is normally not realistic to assume that modem performance will ideally match the theoretical, it is the goal of modem design to come as close to the theoretical plots as possible.

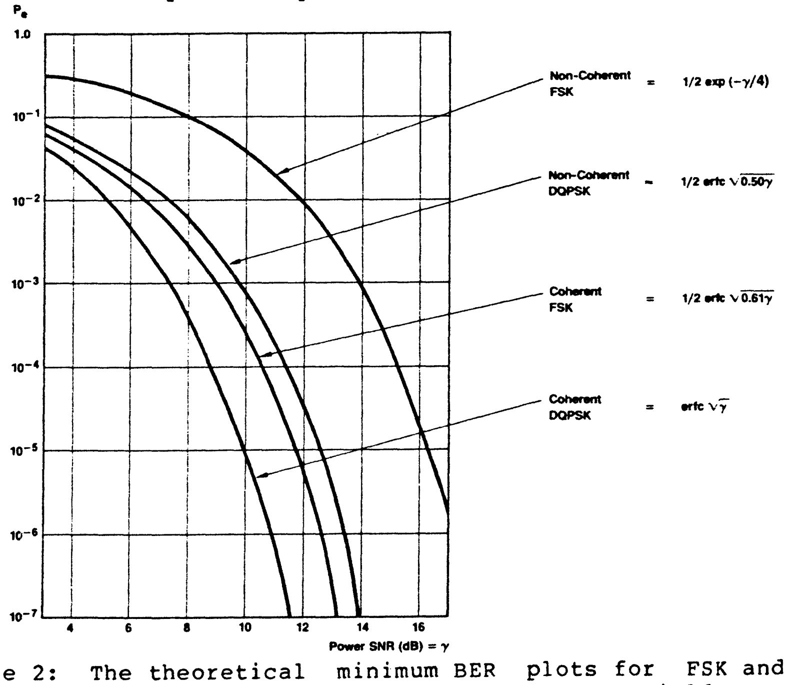


Figure 2: The theoretical minimum BER plots for FSK and DQPSK modems show that coherent demodulators yield superior performance to their non-coherent counterparts.

The Single-Chip Modem Transmitter

There are three basic components to both the FSK and DQPSK transmitter. First, a sine wave synthesizer creates a digital representation of the modulated signal. Because the synthesis is done in a sampled manner and the analog signal is modulated by a digital input, high-level harmonic tones are generated that exceed the Federal Communications Commission (FCC) requirements for transmission on the telephone network. The second block of the modem transmitter is a digital band-pass filter for out-of-band energy. The final portion of the modem transmitter is an on-chip digital-to-analog converter (DAC) that puts a modulated analog signal out to the network protection circuitry.

A problem exists with this method of modulation as it has been stated since the output of the DAC creates harmonic frequency components that interfere with the out-of-band energy specifications mentioned before. To eliminate the harmonics, a steep analog filter could be attached to the output of the transmitter, but these filters are difficult to integrate into silicon. A more attractive solution is to increase the sample rate from the basic value (near 8 kHz) to a far higher value. The sampling rate can be increased in stages, each stage performing a filtering function known as interpolation. Then a single pole analog filter which can be integrated into silicon is attached to the output of the DAC.

The modulated signal is synthesized using a phase-to-amplitude technique which is completely digital. A block diagram of the sine synthesizer is shown in Figure 3. The basic concept is to accumulate phase at the frequency to be generated and then to convert the phase to an amplitude using a ROM look-up.(Ref. 5) This method is quite simple to implement because the non-linear amplitude is created from the linearly accumulated phase. Constants stored in a second small ROM determine the increment of phase added to the phase accumulator.

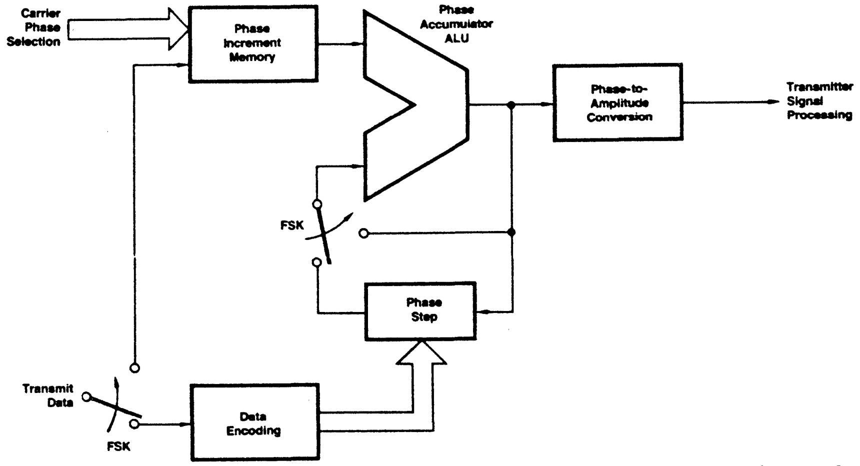


Figure 3: A block diagram of the sine synthesis portion of the transmitter. A phase increment is selected from memory and accumulated to give the absolute phase. The absolute phase addresses a phase-to-amplitude memory.

To avoid a large phase accumulator and sine ROM, and to accurately determine changes in the digital input data, the sine synthesis is performed at a high sampling rate. The sample rate is then decimated down to the lower basic value in several stages. Each decimator stage filters out the energy created by the sampled system at frequencies near the reduced sample rate. Both the decimators and interpolators can be implemented as simple linear-phase FIR filters. The decimators and interpolators for the DQPSK modem can be exactly the same as their FSK counterparts.

Like the decimators and interpolators, the band-pass filters employed on the 1200 bps and the 300 bps modem types have an identical structure. Only the filter coefficients are altered for the two different types of modems. These band-pass filters are IIR filters that are group delay equalized to eliminate phase distortion.

A complete transmitter block diagram is shown in Figure 4. To summarize, the modem transmitter consists of a high sample rate phase-to-amplitude sine synthesis followed by decimation to the basic sample rate. At the lower sample rate, the digitally generated sine wave is band-pass filtered to reduce out-of-band energy. The modulated and filtered signal is interpolated up to a high sample rate again before being presented to the transmitter DAC. The analog waveform has a simple one-pole low-pass filter applied to it to reduce harmonics created by the sampled system.

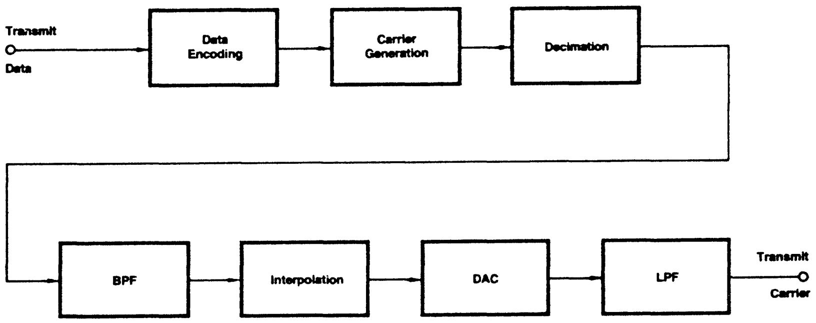


Figure 4: Both the FSK and DQPSK modems are aole to use the same transmitter structure as shown in this diagram. Different coefficients are selected for the band-pass filter section depending on the modem type selected.

The Single-Chip Modem Receiver

The modem receiver demands far more complicated signal processing than the modem transmitter. In the reception process, a distorted analog waveform with an amplitude spanning several orders of magnitude must be converted to a digital pattern having a minmal number of errors. Additionally, the distortion and amplitude of the analog signal can vary from telephone connection to telephone connection. Compare this to a simple frequency synthesis and filtering function for the transmitter and it is completely clear why the modem receiver is more complicated.

The modem receiver consists of a band-pass filter, an automatic gain control (AGC) stage, a demodulator, and in the case of the DQPSK modem a timing recovery stage. The modem receiver system is shown in Figure 5. The analog-to-digital converter (ADC) employed on the single-chip modem uses a high sample rate to alleviate multi-order external analog filters. A single-pole, on-chip analog filter is sufficient to attenuate any high frequencies that might alias back through the ADC into the sampled system. The over-sampled digital waveform produced by the ADC must be decimated down to the basic sample rate of the system. This decimation is done in stages similar to the transmitter decimation, with the decimating filters being linear-phase FIR filters.

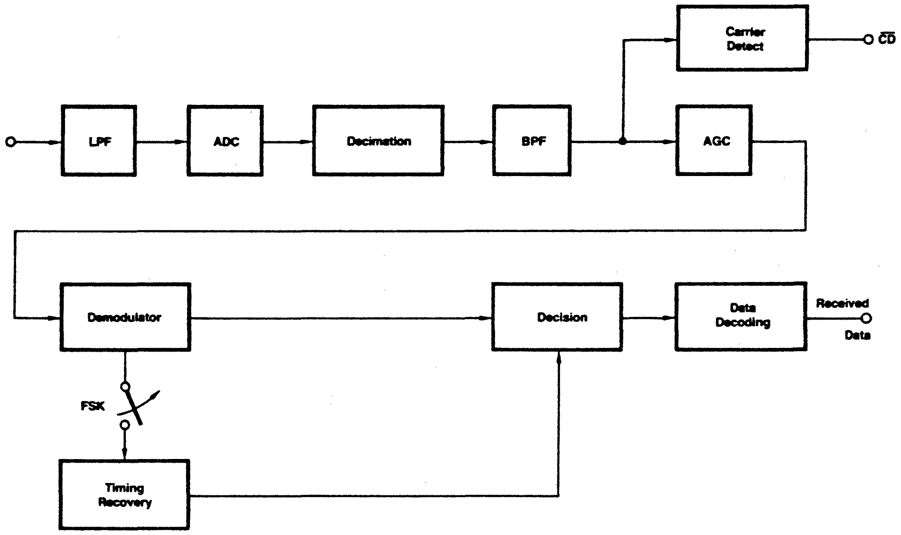


Figure 5: Although the FSK and DQPSK demodulators are not the same, it is possible to execute the same microcode in both cases by carefully choosing coefficients. As in the transmitter, the coefficients differ between the FSK and DQPSK modem types.

Because the public switched telephone network is a two-wire system, and the modem is operating in full-duplex, there is crosstalk from the local modem transmitter into the local modem receiver. The crosstalk frequency spectrum must be band-pass filtered from the receiver input before demodulation can occur. The band-pass filters employed on the single-chip modem are IIR filters that are group delay equalized to reduce phase distortion.

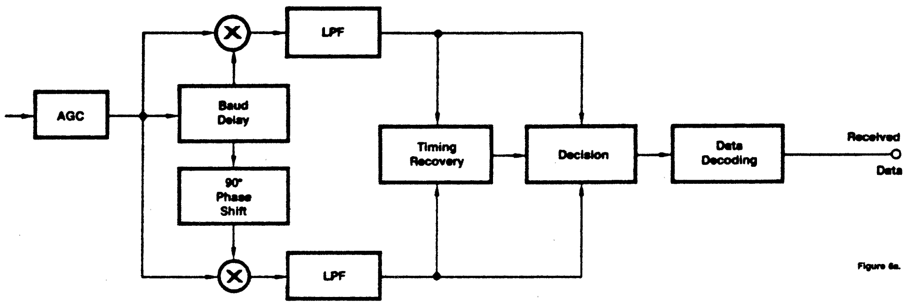
Both the FSK and DQPSK demodulators are quite sensitive to input signal amplitude, thus an automatic gain control (AGC) is used to hold the steady-state amplitude constant. Most AGC algorithms use a feedback technique to derive a fixed-level multiplier from the band-pass filter output. Because the modem is built using DSP, it is possible for the AGC algorithm to be a finite impulse machine without any feedback. The finite impulse algorithm is of great interest because it is physically impossible in an analog system and it offers greater stability than the infinite impulse (feedback) system.

The first step in the AGC algorithm is to develop a power signal from the band-pass filter output. This power signal can be generated in a variety of ways, the most common being a squaring operation or a rectification. The power is next low-pass filtered to eliminate signal variations due to the modulating frequencies. The result of these operations is a slowly varying DC signal that can be used to select a multiplier constant in either a feedback or feed forward loop. The constant is multiplied by the band-pass filter output signal to yield a fixed-level AGC output.

Demodulation of the modulated carrier into its original baseband components may be performed either coherently or non-coherently. Generally, coherent demodulation extracts a reference carrier from the received signal, removes the noise from the carrier, and multiplies the band-pass filtered signal with the clean reference. A block diagram of a typical coherent demodulator is shown in Figure 6(a). (Ref. 6) Notice that the coherent demodulator is a feedback system. Also note that the demodulation requires two carrier signals 90^{o} out of phase to demodulate the quadrature components present in the DQPSK signal. The multiplication of the received signal and the reference carrier generates two spectral components: one at the baseband rate which contains the data information, and one at twice the modulation frequency which contains no information. The double frequency term is filtered out by the low-pass filter following the multiplication. The output of the low-pass filter is then sampled at tne desired location in time by the decision block as a function of the timing recovery circuitry. The decision block provides a dibit output which represents the demodulated data during the current baud interval.

Non-coherent demodulation normally does not extract a clean reference carrier in the same sense as does coherent demodulation. A block diagram of a typical non-coherent DQPSK demodulator is shown in Figure 6(b). (Ref. 3) Notice in this case that the band-pass filtered received signal is delayed by a baud interval, then is divided into two paths which are 90^{o} out of phase. The signal processing for the coherent and non-coherent schemes is quite similar except for the establishment of the demodulating signal. Coherent demodulation provides 2.5 dB greater signal-to-noise ratio performance than non-coherent demodulation. This is explained rather simply by recalling that the coherent demodulator provides a noise-free reference signal for demodulation. The non-coherent demodulation signal still contains in-band noise which has not been filtered out. Thus, in the non-coherent scheme, the carrier is being demodulated with a noisy reference.

Similar to DQPSK demodulation, FSK demodulation may also be performed either coherently or non-coherently. Because FSK modulation is encoding only a single bit per baud interval, the FSK demodulator requires only a single demodulation path. An example of a coherent FSK demodulator using a phase-locked loop is shown in Figure 7(a). Note the similarity of this demodulation scheme to that discussed previously for DQPSK demodulation. The differences are that the loop parameters are



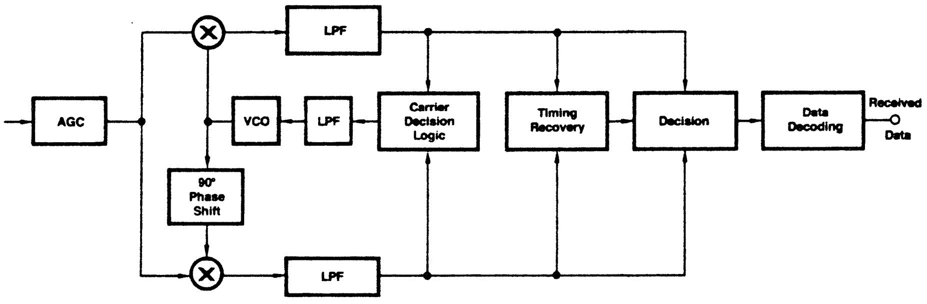
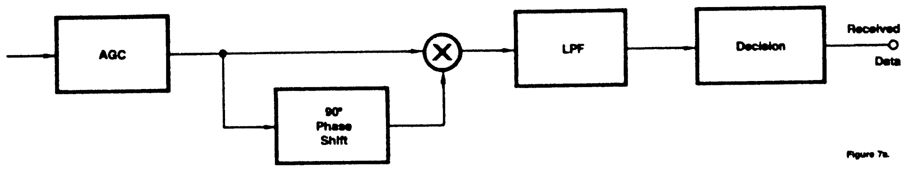


Figure 6: While the non-coherent DQPSK demodulator shown in (a) looks similar to the coherent DQPSK demodulator shown in (b), the carrier extraction in (b) yields superior performance. The coherent demodulator is also known as a Costas loop.

different for the two schemes, and certain blocks are absent in the FSK demodulator, but it is really a "subset" of the coherent DQPSK demodulator. In the coherent DQPSK demodulator, the VCO output is required to generate a stable carrier reference which does not vary with the data pattern and the noise of the received signal. In the coherent FSK demodulator, the VCO frequency should track the received frequency, and not lock to the center frequency between the two frequency shifts. The indication of the data is the output of the loop filter which is then low-pass filtered. The decision block makes its decision on the output of the loop filter as to the content of the data during that baud interval.

Non-coherent FSK demodulation may be performed in a number of different ways. A scheme which maintains compatibility with that required for non-coherent DQPSK demodulation is shown in Figure 7(b). This demodulation scheme is known as a product demodulator. The phase shifter provides a 90° phase shift at the mean frequency between the two frequency shifts. This signal then is used to demodulate the received signal. The key to this demodulation scheme is the phase shift at the FSK mean frequency. If the instantaneous received frequency is above or below the mean frequency, the phase shift will not be 90°, but above or below 90°. After low-pass filtering, the output will provide the cosine of an angle which is either above or below 90°. If above 90°, the cosine is negative; if less than 90°, the cosine is positive. Hence, the decision logic simply tests the sign of the signal from the loop filter for an indication of the bit received during the current baud interval.



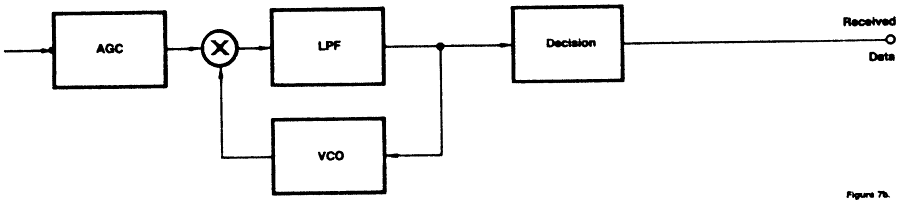


Figure 7: Again, the FSK demodulators look quite similar in nature, but the non-coherent (a) does not perform as well as the coherent (b).

Timing recovery is a function required exclusively for synchronous (DQPSK) modems. The DQPSK modem always transmits data over the telephone network synchronously at 1200 bps. The user may provide data to the modem at character asynchronous intervals (and bit asynchronous, within limits), but this asynchronous data is converted from its asynchronous format and transmitted synchronously over the line. This contrasts with FSK modulation where the user may transmit data through the modem at asynchronous intervals. The FSK carrier will change as a function of the digital data change provided by the user.

Because of the synchronous transmission nature of the DQPSK modem, and the cosine shaping provided in the modem transmitter, there exists an ideal sampling instant where inter-symbol interference (ISI) is minimum (ideally zero). The timing recovery clock extracts the 600-baud clock from the received data. The decision block samples the output of the low-pass filters at synchronous zero-ISI intervals determined by the timing recovery clock. It turns out that the spectrum of the received data provides a spectral null at the 600Hz component required by the timing recovery circuitry. The trick then is to perform a non-linear transformation on the data pattern which will create a spectral component at 600Hz which may then be extracted and used as the timing recovery clock. The system shown in Figure 8 provides this function. One channel of the demodulated baseband data is delayed and multiplied by itself.

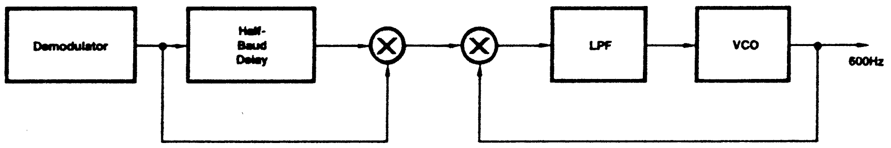


Figure 8: The timing recovery for a DQPSK modem involves creating a double frequency carrier from the demodulator. A phase-locked loop then provides a precise timing signal.

This creates a non-linearity which generates a discrete spectral line at 6ØØHz. This component may then be locked on by a phase-locked loop which provides a clean version of the timing clock.

Algorithm Implementation

The IIR filter, FIR filter, and multiplication algorithms implemented on this single-chip modem have been tailored to work in the shift-and-add architecture of the silicon. Some hardware modifications to the modem ALU are necessary to perform particularly difficult portions of the multpication, AGC and VCO algorithms. The coding of the IIR and FIR filters will first be detailed, followed by some discussion of the multiplication and VCO coding with the requisite hardware changes.

In a standard shift-and-add architecture, an eight-bit filter coefficient requires eight ALU operations per coefficient. This is not only terribly wasteful, but would make the modem implementation virtually impossible in silicon. Fortunately, the architecture allows both add and subtract operations. By grouping the coefficient bits together, it is possible use the addition and subtraction to represent each coefficient in a canonic sign-digit format. An example of a canonic sign-digit coefficient is given below.

$-0.1111001111_2 = -0.9521484_{10}$ binary representation
eight operatons in a add only architecture

$-1.000\bar{1}01000\bar{1} = -2^{0}\,(1-2^{-4}\,(1-2^{-2}\,(1-2^{-4})))$
canonic sign-digit representation
four operations in an add/subtract arch.

($\bar{1}$ means subtraction of this bit instead of addition)

With this enhancement to the standard add-and-shift ALU, it is possible to represent coefficients in both IIR and FIR filters using only three to four bits while incurring a less than Ø.1 % coefficient rounding error. Coefficients greater than 1.Ø in magnitude use a four-bit representation, while those less than 1.Ø are three-bit representations. A standard second-order sectons can be implemented in 17 operations.

IIR filters are designed using a proprietary algorithm developed explicitly for this type of ALU architecture. After the initial IIR filter is designed and the coefficients for the filter are rounded, the filter is conditioned to reduce the possibilty of ALU overflow. The poles and zeros are grouped such that they are within close proximity of one another. This minimizes the amplitude peaks in every second-order section. Next, the filter sections are ordered from lowest to highest amplitude peak (Ref. 7) to provide the least possibility of overflow.

Most of the FIR filters are the simple-coefficient decimators that are quite simple to design and generally use only one or two bits per coefficient. There are a few filters in the modem which are high-order FIR sections. The McClellan-Parks algorithm (Ref. 8) is used to design these filters. Coefficients for the FIR filters are normally rounded to three bits in the canonic sign-digit format.

Multiplications of two data values are required in the AGC, demodulator, and timing recovery blocks of the modem receiver. A modified Booth algorithm was employed to do the multiplication in the shift-and-add architecture. Special hardware was developed in the ALU to take the magnitude of both multiplicands, and to readjust the sign of the product at the completion of the multiplication. After the magnitude of both operands is taken, another section of ALU hardware takes control of the adder/shifter and performs the actual Booth multiplication. When the complete product has been formed, the sign is corrected by another piece of hardware.

The voltage controlled oscillator (VCO) also requires that the ALU hardware be slightly modified for implementation. The sine synthesis portion of the VCO operates on the same principle as the transmitter synthesizer. Two values, the current phase and the current phase increment are stored in the ALU's data RAM. A simple low-pass filter in the phase-locked loop feedback path determines whether to increase or decrease the current phase increment. Increasing or decreasing the phase increment alters the frequency that is being generated by the VCO. The updated phase increment is added to the current phase value before the phase-to-amplitude conversion occurs in the sine ROM.

Conclusion

The single-chip modem described in this paper could be upgraded to work at higher bit rates. An on-chip multiplier is warranted in the silicon implementation of higher speed modems. The multiplier can also be used for the adaptive filtering algorithms necessary to place these modems on the telephone network. Adaptive algorithms would also necessitate some coefficient RAM which was not provided on this Bell 212A modem. The key point to implementing higher speed modems in silicon is the simplicity which DSP provides to upgrade a component. The addition of a multplier and coefficient RAM does not imply a complete redesign of the ALU structure. In fact, the ALU may be used "as is" if the number of microcode instructions has not increased significantly. This concept of upward compatibility cannot be applied to the integrated analog versions of new modem types.

The upward compatibilty of architectures will continue to be applied to build faster modems. The modem described in this paper borrowed its architecture from a previous modem project; as future VLSI modems will borrow from the Bell 212A integrated modem. By using a shift-and-add architecture with some modifications to implement Booth's multplication algorithm, a complete Bell 212A compatible modem has been designed. The canonic sign-digit representation of coefficients and proprietary filter design algorithms help reduce microcode requirements for this VLSI component. Trade-offs between several system block implementations have been considered while trying to maintain microcode compatibility between the FSK and DQPSK modem types. By placing both the FSK and DQPSK modems on a single-chip, this component is a boon for the user who desires full Bell 212A compatibility.

References

1. R.W. Lucky, J. Salz, E.J. Weldon Jr., Principles of Data Communication, McGraw-Hill, New York, 1968, Ch 8.

2. J. G. Proakis, Digital Communications, McGraw-Hill, New York, 1983, Ch 3-4.

3. K. Feher, Digital Communications, Satellite/Earth Station Engineering, Prentice-Hall, Englewood Cliffs, 1981, Ch 4.

4. K. S. Shanmugam, Digital and Analog Communication Systems, John Wiley and Sons, New York, 1979, Ch 8.

5. H. W. Cooper, "Why Complicate Frequency Synthesis?," Electronic Design, pp.80-84, July 19, 1974

6. F. M. Gardner, Phase-Lock Techniques, John Wiley and Sons, New York, 1979, Ch 10.

7. L.B. Jackson, "Roundoff-Noise Analysis for Fixed-Point Digital Filters Realized in Cascade or Parallel Form," I.E.E.E. Transactions on Audio and Electroacoustics, vol. AU-18, pp.107-122, June 1970

8. J.H. McClennan, T.W. Parks, and L.R. Rabiner, "A Computer Program For Designing Optimum FIR Linear Phase Digital Filters," I.E.E.E. Transactions on Audio and Electroacoustics, vol. AU-21, pp.506-526, December 1973

Stevan Eidson was born in Inglewood, California on 4 May 1959. He received his BSCS and BSEE from the University of California, Irvine in 1980 and 1981, respectively. He received an MSEE from UCLA in 1982. From 1980 to 1982, Steve was a member of the technical staff at Hughes Aircraft Company's Ground Systems Group in Fullerton, California. Since 1982, he has been a senior system design engineer responsible for algorithm development in Advanced Micro Devices' Product Planning Division in Sunnyvale, California. Mr. Eidson is a member of Eta Kappa Nu and the IEEE Acoustics, Speech, and Signal Processing Society.

David M. Taylor was born in Sacramento, California on 19 February 1958. He received his BSEE degree from California Polytechnic State University (Cal Poly) in San Luis Obispo in June 1981. He is currently pursuing his masters degree at the University of Santa Clara. Dave was employed by Lockheed Missiles and Space Company in Sunnyvale, California from 1979 to 1981.

Mr. Taylor has been employed by Advanced Micro Devices in Sunnyvale, California since 1981. . His current title is Section Manager in AMD's Product Planning Division. His duties include defining new data communication products and performing the systems design on these products. He has had a number of technical articles published since 1980.

Controller chips pair up in S interface for ISDN systems

Two CMOS LSI chips, the chief building blocks of a system terminal and a PBX line card, simplify implementation of the S interface for ISDN links.

The emergence of all-digital telephone networks using packet- and circuit-switched data schemes will open the door to many new telecommunication services, including electronic mail, banking, alarm signaling, and the transfer of information via telex and facsimile facilities.

Already the International Telephony and Telegraphy Consultative Committee (CCITT) has recommended standards for an S interface on the premises of subscribers to the Integrated Services Digital Network (see "Interface Standard Ensures Compact Subscriber Networks," p. 158). This microprocessor-controlled interface to the ISDN will give subscribers simultaneous access to two 64-kbit/s voice or data channels and one 16-kbit/s control channel. Voice-band signals are to be digitized at the subscriber's terminal and transmitted over one of the 64-kbit/s channels. It will be possible to link all three channels to a main computer for central handling over pulse-code-modulated (PCM) highways.

Alan Clark, Advanced Micro Devices Inc.
Alan Clark, currently ISDN product marketing manager at AMD in Sunnyvale, Calif., joined the company in March 1983 as a development engineer. Before that, he designed microprocessor-based control systems for power stations at Babcock-Bristol Ltd. in the United Kingdom. He holds a BSc in mathematics and physics and a PhD in digital signal processing from the Polytechnic of Central London.

Although it will be some time before system specifications are completely defined, help in implementing the S interface is at hand in the form of two CMOS chips: the Am7930 digital subscriber controller and the Am7931 digital exchange controller. They are the first LSI circuits to meet the CCITT's basic design criteria for ISDN terminals and private branch exchange (PBX) line cards.

The subscriber controller chip is the heart of either a basic or an advanced ISDN terminal driven by an 8-bit microprocessor; it gives the customer access to the data-handling facilities on the S interface. For PBX design, the exchange controller chip is used to build the required number of S interfaces on the line card (one chip for each interface). Residing on the PBX line card itself, the device provides the exchange information function for one ISDN subscriber line.

Both chips allow full-duplex data transfer at 192 kbits/s over four wires between the subscriber's terminal equipment and the PBX line card. These ICs support the point-to-point and point-to-multipoint connections described in the CCITT recommendations, with the data framed on the 64-kbit/s voice and data (B1 and B2) channels and the 16-kbit/s control and data (D) channel.

The main blocks of the subscriber controller chip are a line interface unit, a data link controller, a multiplexer, a main audio processor, and a microprocessor interface (Fig. 1a).

The line interface is connected to the four-wire S interface by a pair of isolating transformers, one for transmission and the other for receiving. The transmitter, which has a binary-to-pseudoternary encoder and a line driver, receives the B channel data from the multiplexer and the D channel data from the data link controller. It then couples these signals to the S interface.

The unit's receiver recovers data and timing from the S interface; it also presents data from the B channels (via the multiplexer) and the D channel (via the data link controller) to the subscriber. The receiver, in essence performing a function inverse to that of the transmitter, converts pseudoternary-coded signals into binary before delivering them to other blocks within the IC.

Data processing is coordinated by the data

Interface standard ensures compact subscriber networks

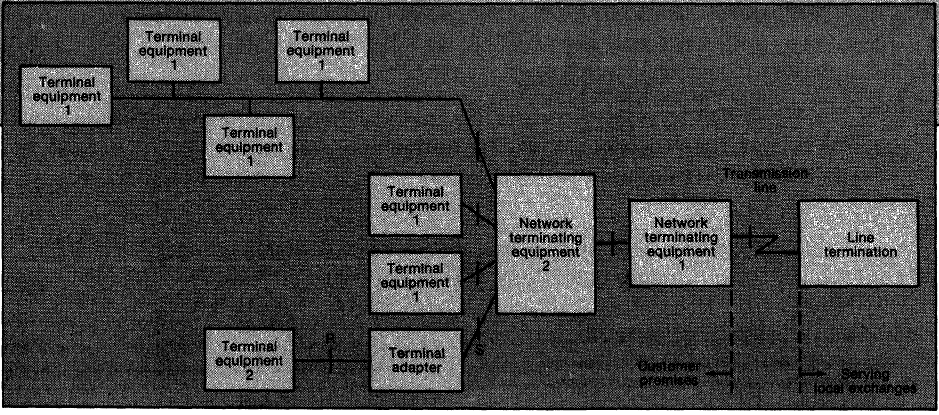
The CCITT's proposals for an interface standard ensure that a small set of ICs can meet the requirements of an evolving telecom network as it moves toward an all-digital state. Two main subscriber configurations—point-to-point and point-to-multipoint—have been recommended for use with the S interface, which is a four-wire module operating at 192 kbits/s (48-kbit data/control frame structure).

In the point-to-point setup, one telephone or terminal is connected via an S interface to a line card in the local exchange through cables no longer than 1 km. The point-to-multipoint configuration may join up to eight telephones or terminals with one S interface, and cable runs to the local exchange are 150 meters or less.

The function of the terminal equipment is virtually identical in both configurations. With the network terminating equipment, however, there is a difference; whereas some equipment would be tied into a single terminal on the subscriber's premises, in larger installations it would be in PBXs using multiple terminals.

In the latter case, network terminating equipment 1 (see the figure) would connect to the transmission line on one side and present the standard S interface at point T on the other side. Network terminating equipment 2 would be optional and would provide line termination for network terminating equipment 1. Terminals that operate with existing standards (like the RS-232) would attach at reference point R and use a terminal adapter for interface buffering.

The transmission line (for which there is yet no standard) would connect either to the line terminator or to a central office line card. The transmission line media, which vary from country to country, could be two-wire, four-wire, fiber-optic, microwave or satellite linkups.



link controller, which extracts the D channel information from the S interface and transmits the microprocessor-generated D channel information over the S interface. The controller also performs bit checking and frame checking for error detection, and it generates various detection flags when necessary.

In operation, the system's master, or external, microprocessor initializes the data link controller and performs the higher-level protocol processing. A random-number generator in the data link controller provides an address for the subscriber controller chip when the IC is powered up for operation on the S interface. When the chip is receiving, data flows via the D channel from the line interface unit through the microprocessor interface. This flow reverses when the chip is transmitting.

The device's protocol permits terminals in the point-to-multipoint configuration to access the D channel in an orderly way. When two or more terminals attempt to access the D channel simultaneously, one will always be successful in completing transmission of its information. This procedure also permits the terminals to operate in a point-to-point configuration.

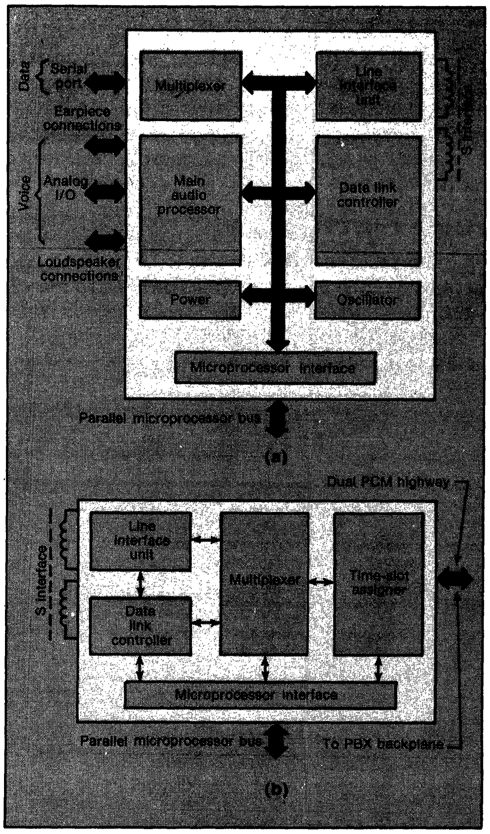
Selective response

The data link controller will recognize 1- or 2-byte addresses and the first 2 bytes in a longer address. This allows the user to program the chip to respond only to selected addresses. In this way the terminals interrupt their respective microprocessor only when their preprogrammed address is received. The data link controller can also be programmed to bypass the address-decoding phase and to pass the entire packet to the microprocessor.

As the data link controller receives the D channel information from the line interface unit, it alerts the external microprocessor by generating suitable interrupts (if there is no interrupt mask). The microprocessor determines the source of an interrupt by reading the interrupt register, and it then accesses the appropriate error or status registers.

The circuitry to access the subscriber controller's programmable, status, and error registers, as well as its transmitting and receiving buffers, is all in the microprocessor interface block. This section has the logic to interface the chip to an external 8-bit microprocessor. Controlling the multiplexing of the B1 and B2 channels to and from different sources is the job of the programmable multiplexer. Possible source and destination pairings include the line interface unit, the main audio processor, the microprocessor interface, and the serial port, with the port providing a serial interface to external peripherals.

Generating suitable audio responses and



1. The Am7930 digital subscriber controller for telephone systems contains circuitry for accessing the S interface defined by the CCITT (a). This device is on the subscriber's premises. The Am7931 digital exchange controller provides the S interface and line termination functions for the subscriber controller (b). The IC is on the private branch exchange (PBX) line card.

tones is the responsibility of the main audio processor block. In addition to digital filtering circuits, it has digital-to-analog and analog-to-digital converters. It will drive a handset earpiece or a loudspeaker, and its analog input accepts a handset mouthpiece, a separate microphone, or both.

The multitone generator in this block can provide a one- or two-tone signal with programmable frequency, amplitude, and cadence. One or more tones can be summed into the transmission (and side-tone) paths to create a dual-tone multifrequency generator. Or the tones can be summed into the receiving path to provide indication at the loudspeaker output of dialing, busy, or ring-back conditions.

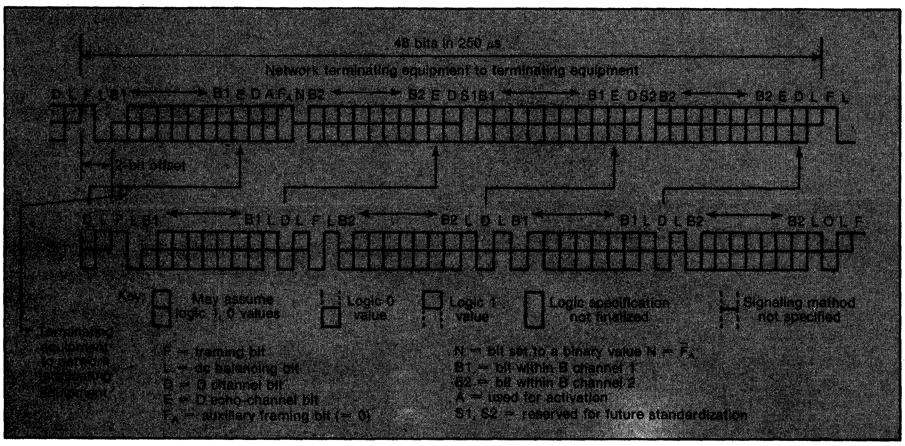
Two correction filters, one in each transmission and receiving path, can be programmed to modify the frequency characteristics. Thus they compensate for any imperfections in audio response of the microphone, earpiece speaker or loudspeaker. They can also be used to add preemphasis or postemphasis or both to match user preferences.

Except for the exclusion of the main audio processor and the inclusion of a time-slot assigner, the main blocks in the exchange controller chip are the same as those of the subscriber controller (Fig. 1b). There are, however, some operational differences within the blocks of the exchange controller.

First, the line interface unit can be operated in two modes: adaptive timing for point-to-point applications and fixed timing for point-to-multipoint. The mode is selected by an internal register that the local external microprocessor programs.

Also, the exchange controller's multiplexer can route data from the microprocessor interface and the B1, B2, and D channels to the time-slot assigner. The multiplexer performs these operations in addition to the routing options specified for the multiplexer in the subscriber controller chip. When the D channel is routed to the time-slot assigner, it bypasses the data link controller. Hence the central computer has direct access, over the dual PCM highway, to the B1, B2, and D channels to or from the S interface, as well as to the local external microprocessor.

Significantly, the exchange controller chip is compatible with the Am7901A subscriber line


48 bits in 250 μs
Network terminating equipment to terminating equipment
D L F L B1 B1 E D A F_A N B2 B2 E D S1 B1 B1 E D S2 B2 B2 E D L F L
2-bit offset
D L F L B1 B1 L D L F L B2 B2 L D L B1 B1 L D L B2 B2 L O L F
Terminating equipment to network terminating equipment
Key:
May assume logic 1, 0 values
Logic 0 value
Logic 1 value
Logic specification not finalized
Signaling method not specified
F = framing bit
L = dc balancing bit
D = D channel bit
E = D echo-channel bit
F_A = auxiliary framing bit (= 0)
N = bit set to a binary value N = F̄_A
B1 = bit within B channel 1
B2 = bit within B channel 2
A = used for activation
S1, S2 = reserved for future standardization

2. The data frame structure with which the subscriber and exchange controllers communicate over the S interface operates at 192 kbits/s. During each 48-bit (250-μs) frame, two voice/data streams and one control/data stream are sent from the network terminating equipment to the terminal equipment. The identical format controls communications between terminal equipment and network terminating equipment after a check for line contention has been performed by the terminal equipment (D to E bit query). The CCITT ISDN recommendation gives a full definition of frame structure and bit polarities.

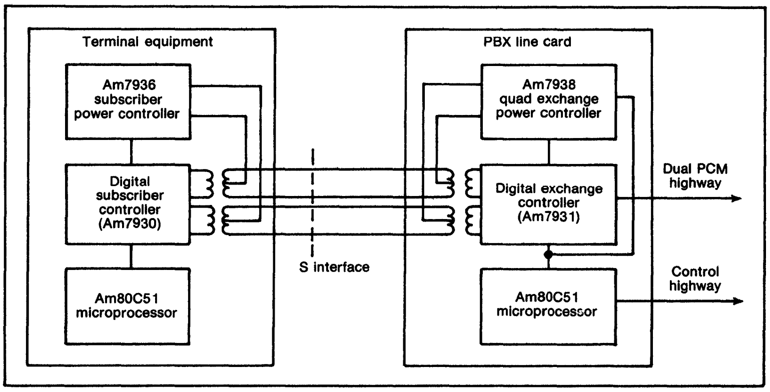
audio-processing circuit (SLAC) used in analog line cards, because both parts support a dual PCM interface. Therefore digital line cards can replace analog line cards on a one-to-one basis within the same PBX. The practical advantage is that an analog system can be gradually modified to an all-digital one. Most importantly, because of the backplane compatibility between the exchange controller chip and the subscriber line audio processing circuit (vis-à-vis the dual PCM highway), systems can be developed that connect the S interface point to analog trunk lines. This configuration befits public telephone communication.

Picking the right slot

The exchange controller's dual PCM interface has independent control of the transmission and receiving paths. The frame synchronization signal identifies the beginning of the frame and establishes a reference point for all time slots. The dual PCM interface can operate at up to 8.192 Mbits/s and therefore will be able to accommodate 256 time slots at 64 kbits/s. During each frame, two 64-kbit/s B channels (8-bit time slots) and one 16-kbit/s D channel (2-bit time slot) may be transferred to or from the dual PCM highway (Fig. 2). Time-slot assignment and highway selection is under program control of the line card microprocessor.

The on-chip time-slot assigner allows a number of the exchange controllers to be connected together and to the PBX backplane. The assigner can be programmed by the microprocessor to transmit or receive data to or from the B or D channels over the PCM highway. Or it can be programmed to send or receive data from the microprocessor. The local microprocessor may thus communicate with the central computer over the dual PCM interface.

When suitably configured with other available chips to create a simple PBX, the subscriber and exchange controller chips can be used in applications ranging from a simple voice telephone to a sophisticated voice/data terminal. In its simplest form, an ISDN terminal using the devices requires a microprocessor and keypad. The microprocessor scans the keyboard and constructs the signaling messages for the D channel. The only other circuits required (Fig. 3) are the Am7936 subscriber power controller and the Am7938 quad exchange power


Terminal equipment
Am7936 subscriber power controller
Digital subscriber controller (Am7930)
Am80C51 microprocessor
S interface
PBX line card
Am7938 quad exchange power controller
Digital exchange controller (Am7931)
Am80C51 microprocessor
Dual PCM highway
Control highway

3. In PBX applications, the minimum set of chips required for subscriber access over the S interface comprises a digital subscriber controller, a digital exchange controller, a subscriber power controller, a quad exchange power controller, and two 8-bit microprocessors. The rudimentary connection shown establishes standard telephone service.

ISDN controller chips

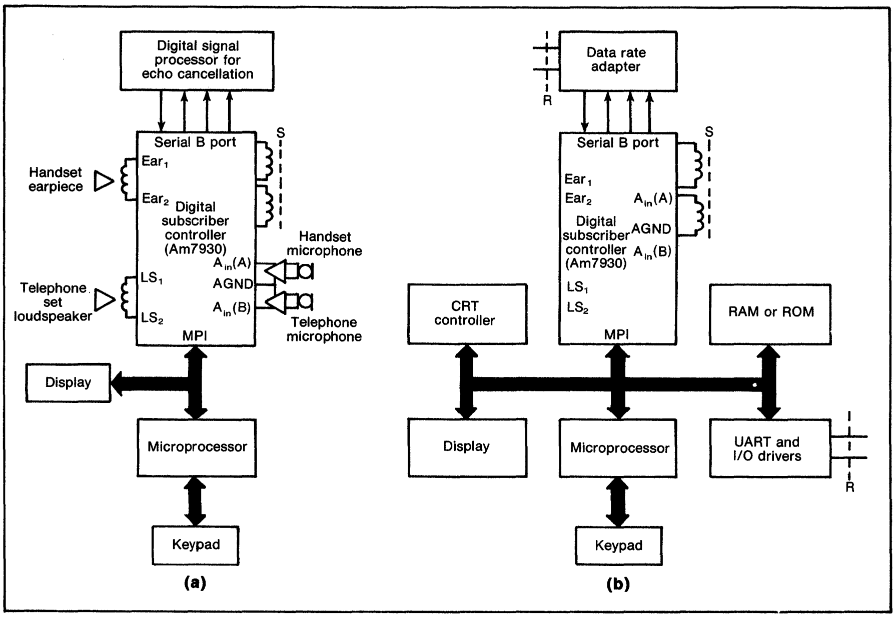
controller, plus two bipolar high-voltage ICs to supply the required voltage and power levels over the S interface.

In the PBX application, a 40-V input to the S interface from the quad exchange power controller is converted by the subscriber power controller into a 5-V dc output, enough to power the ICs in the terminal equipment. One quad exchange power controller can drive up to four S interfaces.

The subscriber controller chip can be used in a variety of terminal equipment, from simple voice-only setups and hands-free phones to a full workstation (Fig. 4). To connect ISDN circuits to existing equipment, the chip's serial or microprocessor ports can be used to access the CCITT's R interface point, which can be any interface—for instance, an RS-232.

Multiple exchange controller chips may be used on one PBX line card to provide the required number of S interfaces. One microprocessor can control up to eight S interfaces and can be linked to the line card's backplane via an Am8530 serial communications controller that connects to the parallel microprocessor bus. Or the microprocessor can communicate with the line card over the dual PCM highway of an exchange controller.

For practical implementation of an S interface on the ISDN, consider a telephone capable of supporting one analog line (one two-way conversation) and an RS-232 data link (Fig. 5). Ideally all the ICs would reside in the telephone, with a standard RS-232—compatible connector

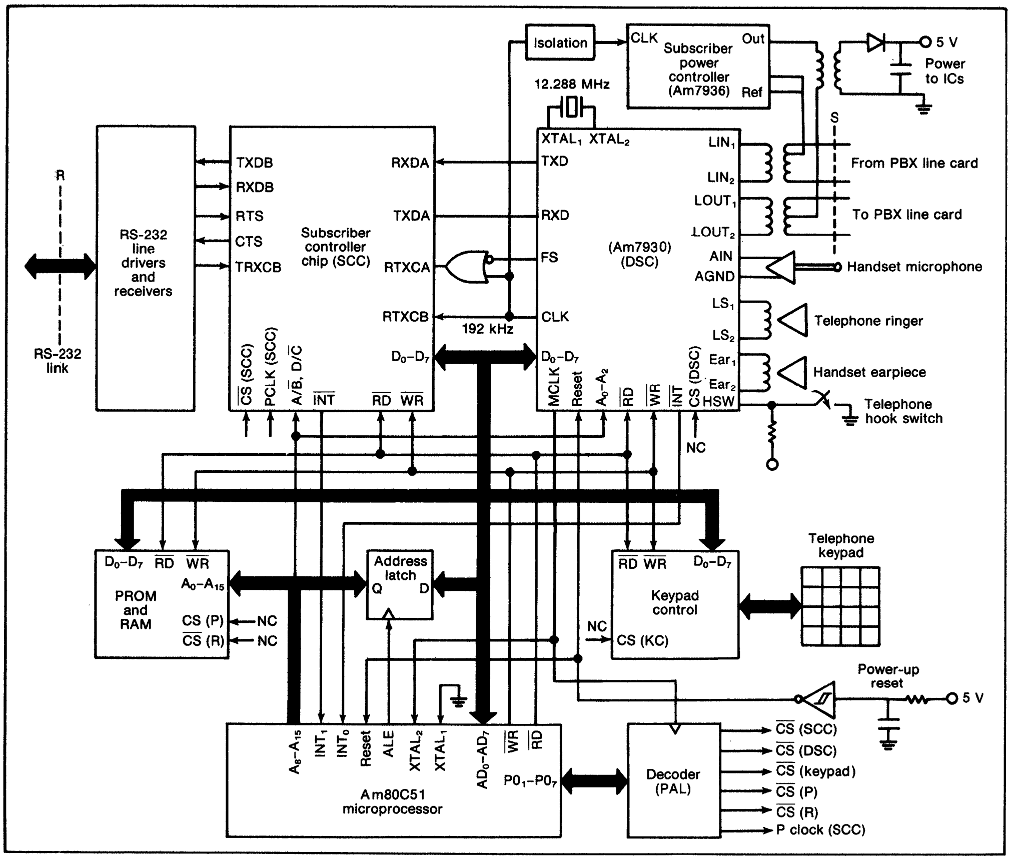


4. The voice-only phone (without the digital signal processor block or a display) is the ISDN equivalent of the standard analog phone. A digital signal processor may be interfaced to the serial B port for echo cancellation and voice encryption, yielding secure hands-free operation (a) with on-hook dialing. In voice and data workstations, the subscriber controller chip combines with a number of peripherals, including memory, UART, CRT controller, and a microprocessor—all connected to the same microprocessor bus (b).

built into the side of the telephone set.

The subscriber power controller provides an isolated 5-V dc output to power ICs required for telephone functions even when local power is lost. The isolation prevents a ground loop between the terminal equipment and the PBX line card when a locally powered terminal is connected to the telephone set.

The microprocessor—in this example, an Am80C51—programs and controls the ICs in the terminal equipment, as instructed by commands generated by the PBX line card, keypad, or RS-232 link. This configuration allows the microprocessor to use its standard 8-bit bus. Port 3 of the microprocessor provides bus control signals (like read and write) and port 0 handles the 8-bit address/data bus. Port 2 handles the high-order addresses (A_8 to A_{15}) to the digital subscriber controller, the serial communications controller, PROM and RAM, and the keypad control. Both interrupt outputs of those controllers tie into the two microprocessor input interrupt pins on port 3, so that the microprocessor can service interrupts from the ICs separately. In addition, with a PAL device, port 1 on the microprocessor delivers the Chip Select



5. Practical terminal equipment applications for controller chips include a telephone that supports one voice line and one data link simultaneously. Memory devices—which should be large enough to accommodate expanded system telephone functions—contain the operating protocols and software for system initialization.

signal ($\overline{\text{CS}}$) to the ICs and a clock to the serial communications controller.

The amount of memory (PROM and RAM) needed depends on the sophistication of the terminal equipment. The PROM contains the LAPD (Link Access Protocol Type D) protocol, keypad, and initialization routines for the subscriber and communications controllers. The number of programs in the PROM—and hence its size—will grow as the local terminals that need the RS-232 link increase and as telephone functions are added to the system (for example, call forwarding, call holding, and automatic redialing). Therefore the PROM area should be upwardly compatible to accept EPROMs of 16 to 64 kbytes. The RAM is used by the microprocessor to store temporary information and to adapt the data rate of the information passed between the RS-232 link and the S interface.

The subscriber controller chip is connected to the telephone earpiece, ringer, microphone, and hookswitch, and, via the S interface to the PBX line card. The serial port on the device connects to one of the serial ports of the serial communications controller. The other serial port on that controller, connected to the differential line drivers and receivers, provides an RS-232 link to a local terminal from the connector on the side of the telephone set.

On power-up, the microprocessor in the terminal equipment enters a reset routine stored in the PROM. This routine initializes the subscriber and communications controllers, then disables them, and the microprocessor then scans the keypad (via the keypad controller) for activity. The terminal equipment can now begin transmitting and receiving to and from the S interface, RS-232 link, and telephone handset. It can be activated by the PBX line card over the S interface or locally, from the keypad or when the handset is lifted.

If the telephone handset is lifted, the subscriber controller chip will interrupt the microprocessor, which interrogates the device's interrupt register and establishes that a voice connection has been requested. The microprocessor then transmits a data packet over the D channel to the PBX, requesting that a B channel be made available for the telephone conversation. (This transmission is over the S interface via the on-chip data link controller.)

At this point the PBX processes the packet and responds by transmitting a corresponding packet from the line card over the S interface to the terminal equipment. The packet indicates which B channel can be used for the voice link, and the packet is passed by the subscriber controller, via its data link controller, to the microprocessor. The microprocessor decodes the packet and programs the chip to connect its main audio processor to the designated B channel on the S interface. With the telephone link thus established, the subscriber will hear the dial tone and be able to dial the telephone number.

The microprocessor scans the keypad for the number being dialed, then transmits it to the PBX on the D channel. The PBX decodes the number and makes the connection. The subscriber will hear the connected telephone ringing, and conversation can begin when the phone is answered via the designated B channel.

The microprocessor and the PBX line card interact similarly when a local terminal is connected to the RS-232 port and the subscriber requests data transmission via the keypad. Unlike voice transmission, however, the microprocessor in this case will program the subscriber controller to route the designated B channel to the serial port, thus providing a link to the serial communications controller. The microprocessor will also program that controller to process the data going to and coming from the subscriber controller's serial port.

A voice from beyond

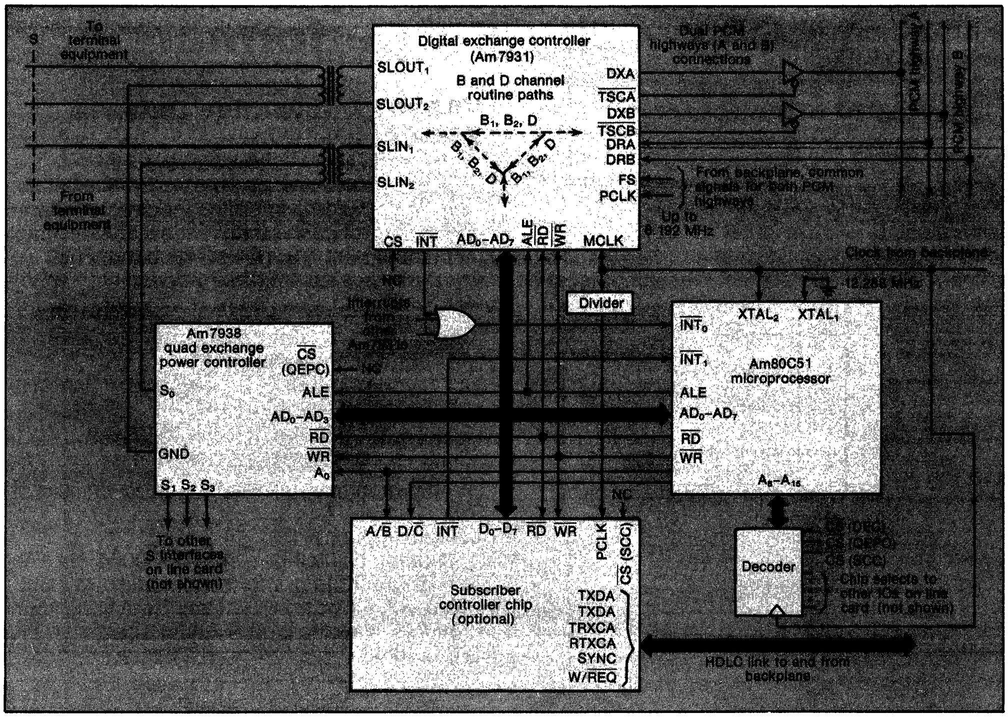
On the other hand, when the PBX initiates a voice transmission, the subscriber controller will interrupt the microprocessor and pass the D channel data from the line card to the microprocessor. If the line card is requesting a voice connection, the microprocessor instructs the subscriber controller to activate the telephone ringer and make the B channel link, so that conversation can begin when the telephone is answered.

If the PBX requests a data link to the RS-232 interface, the microprocessor will program the subscriber controller accordingly, route the designated B channel to the chip's serial port, and monitor the data transfer. Since the S interface can support two 64-kbit/s B channels, voice and data links can be made and run simultaneously.

Whenever a data link is made, from either the subscriber or the PBX line card, the data flows from the local terminal via the RS-232 link through the subscriber controller into the RAM on the microprocessor bus. The data is stored in the RAM and retransmitted on the microprocessor bus through the other serial port on the communications controller. Then it enters the subscriber controller's serial port and passes through the S interface on the designated B channel.

For data coming from the S interface, the data is transmitted through subscriber controller's serial port into the communications controller, then into the RAM via the microprocessor bus. When the data is collected in the RAM, it is retransmitted over the microprocessor bus through the other communications controller serial port, then onto the RS-232 link, and finally to the local terminal.

When the SCC is in the HDLC (High-level Data Link Control) mode, packet data can be transmitted to and received from the local terminal over the S interface via the B channels. Use of the RAM for temporary storage of data coming from and going to the local terminal al-


S
To terminal equipment
From terminal equipment
Digital exchange controller (Am7931)
B and D channel routine paths
SLOUT₁
SLOUT₂
SLIN₁
SLIN₂
B₁, B₂, D
DXA
TSCA
DXB
TSCB
DRA
DRB
FS
PCLK
CS INT AD₀–AD₇ ALE RD WR MCLK
Dual PCM highways (A and B) connections
PCM highway A
PCM highway B
From backplane, common signals for both PCM highways
Up to 8.192 MHz
Clock from backplane
12.288 MHz
NC
Interrupts from other Am7931s
NC
Divider
Am7938 quad exchange power controller
CS (QEPC)
S₀
ALE
AD₀–AD₃
RD
WR
A₀
GND
S₁ S₂ S₃
To other S interfaces on line card (not shown)
INT₀
XTAL₂ XTAL₁
INT₁
Am80C51 microprocessor
ALE
AD₀–AD₇
RD
WR
A₈–A₁₅
NC
A/B D/C INT D₀–D₇ RD WR PCLK CS (SCC)
Subscriber controller chip (optional)
TXDA
TXDA
TRXCA
RTXCA
SYNC
W/REQ
Decoder
CS (DEC)
CS (QEPC)
CS (SCC)
Chip selects to other ICs on line card (not shown)
HDLC link to and from backplane

6. The exchange controller chip is ideal for establishing a PBX, as shown on a section of the line card (only one S interface). The microprocessor would be common to all exchange controllers. The quad exchange power controller can provide power for up to four S interfaces.

lows these operations to occur at different rates. With this capability, data can be transmitted through the system at a fixed 64 kbits/s, required for the B channel, regardless of the rate at which it was originally received.

Extending the highway

Another example shows a portion of a typical PBX line card architecture and the connections and ICs associated with one S interface (Fig. 6). The microprocessor is common to all the exchange controller chips on the line card, and the quad exchange power controller provides the power for up to four S interfaces. The decoder provides the necessary chip select signals.

There are several possible routes for the B and D channels through the exchange controller chip. The B channels from the S interface, for example, can be routed directly to the microprocessor on the line card or over the dual PCM highway via the exchange controlller.

Such a line card configuration is very flexible. When a packet on the D channel is received from the terminal equipment, the exchange controller can pass the packet off the line card directly over the dual PCM highway. Alternatively, it can pass the packet, via its on-chip data link controller, to the microprocessor for processing on the line card.

Conversely, when a packet is transmitted over the S interface to the terminal equipment, it can pass directly to the exchange controller from the PCM highway or from the microprocessor on the line card. In the latter case, the packet is passed through the on-chip data link controller before being transmitted.

The microprocessor can communicate with the host computer controlling all the line cards in the PBX by transmitting and receiving data over the PCM highway (via an exchange controller). As an option, a serial communications controller can be interfaced to the microprocessor bus, so that communication can take place between the line card and the host computer over the controller's HDLC link.□

Data encryption: high speed implementation of DES

A la suite de l'adoption de l'algorithme DES (système de codage de données) d'IBM en tant que norme NBS de codage de données, des circuits intégrés ont été mis au point et permettent und utilisation facile de ce système dans les applications nécessitant la protection des données, pour des raisons de sécurité. Cet article décrit un nouveau circuit intégré qui permet la mise en oeuvre rapide du système de codage des données DES.

Im Anschluß an die Annahme des IBM-DES-Algorithmus als NBS-Norm für die Datenverschlüsselung sind jetzt integrierte Schaltungen eingeführt worden, die eine leichte Benutzung des Systems in Datenschutzanwendungen gestatten. Dieser Artikel beschreibt eine neue intergrierte Schaltung zur DES-Durchführung bei hoher Geschwindigkeit.

Como consecuencia de la adopción por IBM de algoritmos DES como norma NBS para cifrado de datos, se han introducido circuitos integrados para permitir el fácil uso de este sistema para aplicaciones de datos secretos. Este artículo describe un nuevo circuito integrado que realiza operacines DES a gran velocidad.

The growth of electronic data processing applications and electronic communication that involves the transfer of potentially sensitive material between numerous remote sources necessitates a requirement to protect the privacy and security of the material.

A few years ago in the USA, the National Bureau of Standards (NBS) adopted an algorithm from IBM for data security purposes. This algorithm is known as the Data Encryption Standard (DES). The purpose of the standard was to ensure compatibility of hardware of DES devices from different manufacturers.

Principles of the DES algorithm

The DES algorithm operates on 64bits of plain text to produce 64bits of cipher text under the action of a 56bit keying parameter. The 56bit key allows a 72×10^{15} possible combinations. This large number of combinations of keywords makes it difficult for the computer thief to search for the key to decipher data.

To implement DES there are essentially four steps involved:

(i) initial permutation
(ii) key schedule
(iii) cipher function
(iv) inverse permutation.

A block diagram of the DES algorithm is shown in fig. 1.

The 64 bit input block is first passed through an initial permutation (IP). The role of the permutation is to thoroughly mix the data bits and form an 8×8 matrix. This matrix generates a 64bit word. This word is loaded into two registers (L,R), 32bits each. Simultaneously, the 64bit key word goes through a similar permutation. 48bits of the permutation are selected to generate subkey K.

The R-register goes through a cipher function f as shown in fig. 2. This is done by expanding the 32bit R word into a 48bit word and performing a modulo-2 addition with the selected 48bit key (known as key schedule). This sum provides address to S-box (substitution). The output of this look-up table goes through another permutation and generates a 32bit cipher function output.

This 32bit output performs a modulo-2 addition with the contents of the L-register and the output is loaded into the R-register for next iteration. The previous contents of the R-register are then loaded into the L-register. Key scheduling and S-box look-up are repeated sixteen times as shown in figure 1. The final contents of the L- and R-register go through the inverse of the initial permutation to generate a 64bit final ciphered data. For deciphering the process is simply reversed (fig. 4).

The least significant bit of each key byte is

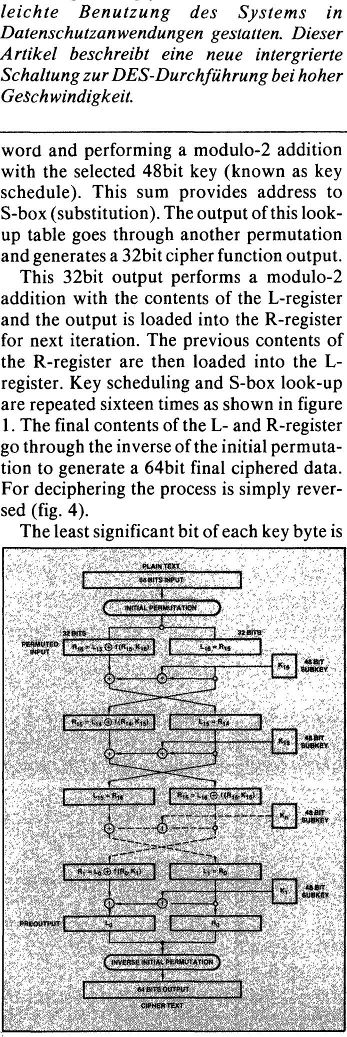


Fig. 1 Data protection has become a prevalent topic in recent years. This diagram shows a flow chart of the DES algorithm

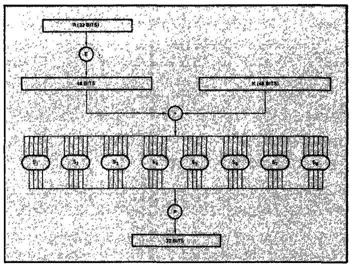


Fig. 2 Showing DES cipher function

the parity indicator for that byte which does not take part in key schedule routine. The remaining 56bits of key word is left-shifted by predetermined amounts and 48bits is selected as the key for cipher function. For decryption, the key word is right-shifted by the same amount. A flow chart of the key schedule is shown in fig. 3.

Selection of encryption methods

The data ciphering processor (DCP) *AmZ8068* from Advanced Micro Devices implements the DES algorithm in hardware in one LSI package and supports three of the enciphering/deciphering methods recommended by the National Bureau of Standards. These are:

- electronic code book (ECB),
- cipher block chaining (CBC), and
- cipher feedback (CFB, 1 byte).

These modes are user programmable. Additionally, the device could be used for 1bit cipher feedback and 64bit output feedback.

The ECB mode is the most basic implementation of DES. It consists of 64bits of input block and 64bits of key DES algorithm to produce 64bits of ciphered output (the flow chart is shown in fig. 5). This process is used mainly for disk and tape applications.

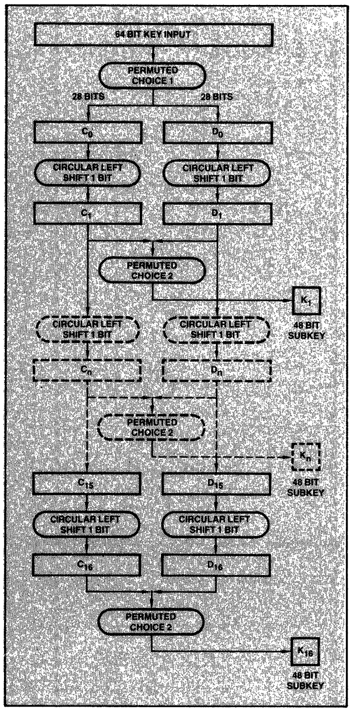


Fig. 3 Key scheduling, as illustrated by the above flow chart, is a major part of DES

CBC operation is accomplished by initially loading a 64bit vector. Each 64bit block input is exclusive-OR'd (X-OR) with the current value of the initial vector register. The result of this (X-OR) is used as the input block to DES. The output of the DES is transmitted and also fed back into the initial vector register for processing the next block of input data (see fig. 6). Since each block is a function of the previous block, the advantage of using CBC is that it secures against message insertion and deletion. This process is primarily used in high speed telecommunication.

The operation of CFB (1 byte) is somewhat similar to CBC. In this case the initial vector register is loaded with a 64bit value. This vector is used as the input block to DES. The most significant 8bits of DES output is (X-OR'd) with 8bits of input data. The result of this (X-OR) is transmitted and also left shifted into the initial vector (the flow chart is shown in fig. 7). This is used mainly for low speed character transmission.

Applying DES

The heart of the DCP is the ciphering algorithm unit (a block diagram of the device is shown in fig. 8). This performs the NBS data encryption which uses a 56bit key (+8bits parity) to encrypt 64bit blocks of clear data ('plain text') into corresponding 64bit blocks of encrypted data ('cipher text'). The DCP has three ciphering keys: one for encryption of clear data, one for decryption of ciphered data and a master key used for generating encrypted keys (session keys).

Both clear and ciphered data passes into the algorithm unit from the 64bit (1 byte in CFB mode) input register and out through the 64bit output register (1 byte in CFB mode). Transfers between these registers and external pins occur on the 8bit I/O busses. Two 8bit wide bidirectional data ports (master and slave ports) are provided for interfacing these busses to external logic. The dual ports, separate busses, and separate input and output registers create a highly pipelined data path which maximizes the throughput by allowing simultaneous input, ciphering and output operation.

One of the bidirectional data ports, the master port, is designed to interface directly to a general purpose μP and almost all the functions of the DCP can be exercised through this port. In particular, command/status registers, mode registers and I/O data registers are directly addressable through this port and encryption/decryption keys may be loaded by command/data sequences.

A second port known as the slave port is used for data I/O operations only.

A third or auxiliary port provides an alternative key entry facility. In particular, the

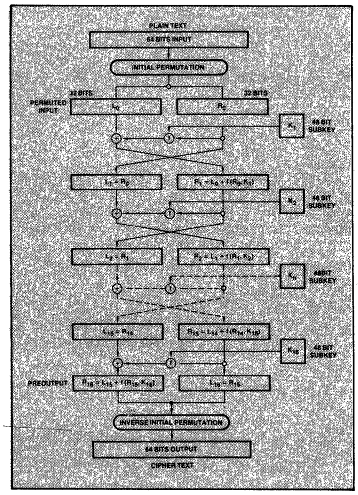


Fig. 4 For deciphering, the data encryption process is simply reversed

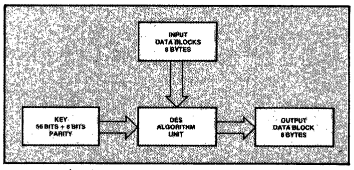

INPUT DATA BLOCKS 8 BYTES
KEY 56 BITS + 8 BITS PARITY
DES ALGORITHM UNIT
OUTPUT DATA BLOCK 8 BYTES

Fig. 5 The electronic code book method, for use in high-speed disk and tape applications

master key register can only be loaded through this auxiliary port. Depending on the mode of operation selected, this port may be used for direct command input and status output for high speed operation. The DCP also has built in parity check circuitry which checks for odd parity on each keybyte.

Modes of operation

The *AmZ8068* has two different modes of operation — multiplexed control mode of operation and direct control mode of operation. Either one of these operational modes is

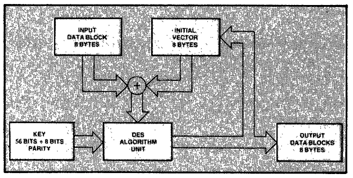


Fig. 6 Showing the cipher block chaining method, for use in high-speed telecommunications

selected by the level on the C/$\overline{K}$ pin. In multiplexed control mode (C/$\overline{K}$ = 0) all the commands and the command data is entered through the master port. The key entry can be achieved through the master port or auxiliary port; and data entry and removal can be done through the master or slave port. This mode of operation is intended for controlling the DCP from a microprocessor.

In direct control (C/$\overline{K}$ = 1) all the control of the DCP is provided at the auxiliary port. This mode of operation is used when the DCP is being controlled from a bit-slice or sequencer controller and allows a higher throughput rate.

continued on page 36

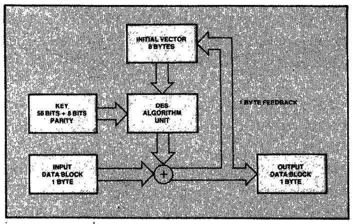


Fig. 7 The cipher feedback mode, for use in low-speed character transmission applications

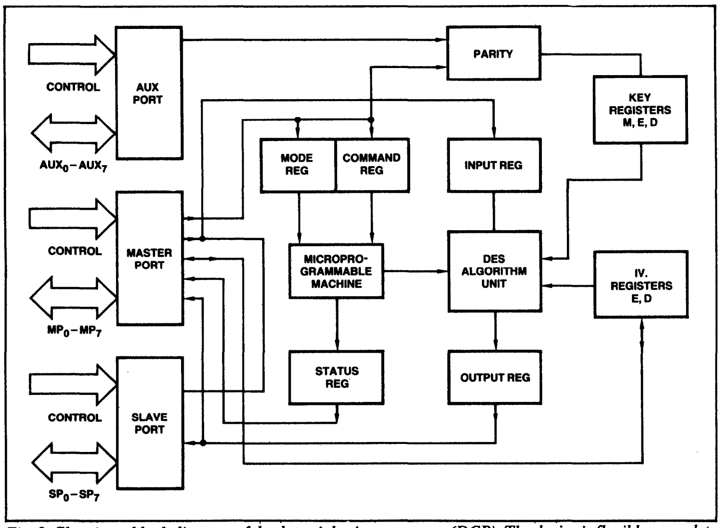


Fig. 8 Showing a block diagram of the data ciphering processor (DCP). The device is flexible enough to be interfaced with a μP bus or sequencer

continued from page 35

Application of the data ciphering processor

The DCP is capable of running at a clock rate of 4MHz. To achieve maximum throughput, the device is operated in dual port mode. For example, one port (master port) deals with input and one port (slave port) deals with output. The data flow in the device is fully pipelined so that simultaneous reading of encrypted data, processing of clear data and writing clear data is possible, resulting in an eighteen clock encryption cycle of 64bit blocks. To obtain a throughput rate of 1,7Mbytes/second the device is operated in ECB or CBC mode, and this is fast enough for most disk controllers and communication channels.

The data ciphering processor architecture is flexible enough to connect to a microprocessor bus or, alternatively, can be operated by a sequencer. A system diagram of how to interface the DCP to a μP is illustrated in figure 8.

Subodh K. Banerjee
Advanced Micro Devices

Data-compression IC saves on storage and speeds up image transmission

An integrated circuit that implements two compression standards brings high-resolution image transmission systems one step closer

by James Williamson and Shinkyo Kaku, *Advanced Micro Devices Inc., Sunnyvale, Calif.*

□ As the costs of magnetic and optical mass storage devices decrease—and as buffer capacities provided by dynamic random-access memory technology increase—digitized image data can be stored and manipulated more economically than ever before. Digital image formats provide very effective means to enter, edit, and transmit image data to remote systems.

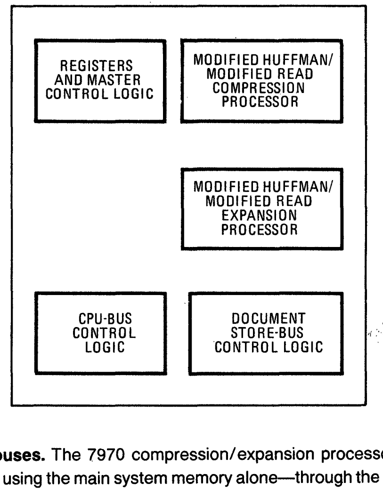
Despite this trend, pixel image representation still remains relatively expensive. For example, an 8½-by-11-in. image scanned at a moderate resolution of 300 by 300 pixels/in. requires over 1 megabyte of storage per image. Clearly, the system storage requirements for multiple images can become prohibitive. Further, transmitting one page of this image data over telephone lines at 9,600 bits/s takes about 15 minutes—and higher-resolution images eat up even more transmission time.

There are alternatives for lowering the storage requirements and the transmission times in a system design: reducing the document scan resolution, or converting existing images to a lower resolution; increasing the communication rate; or compressing the images by removing redundancies. Because scan resolution is critical to the legibility of an image, resolution requirements are generally built into the original system requirements by quality constraints.

Transmission factors

Transmission rates vary widely, depending on such factors as carrier medium, error immunity, and protocols, but these factors are cost-limited in systems that need to communicate with multivendor equipment. On the other hand, because the essence of data compression is the efficient encoding of data to minimize redundancy, compression of the images by removing redundancies is the alternative that can greatly enhance the storage economy and also decrease transmission times in any two-tone, image-based system.

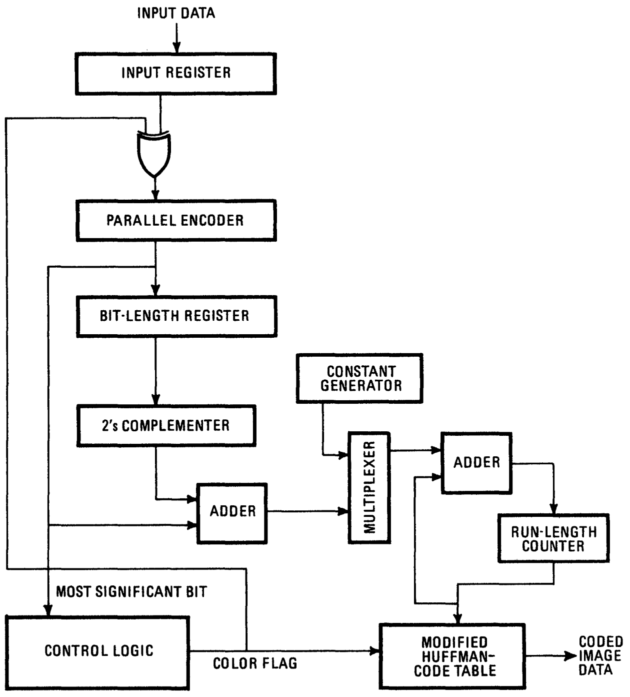
One consequence of redundancies in an image is that too much storage space, or transmission time, is wasted on the correlative data when a bit map is stored or transmitted. Compression reduces these redundancies, multiplying the effective data densities. Because image data usually contains significant redundancies, incorporating even a modest 10:1 compression ratio reduces the 1-megabyte/page example to a more manageable 100 kilobytes/page. This data density corresponds to a transmission time of 1.5 min at 9,600 b/s—a net tenfold increase in transmission speed.


REGISTERS AND MASTER CONTROL LOGIC
MODIFIED HUFFMAN/ MODIFIED READ COMPRESSION PROCESSOR
MODIFIED HUFFMAN/ MODIFIED READ EXPANSION PROCESSOR
CPU-BUS CONTROL LOGIC
DOCUMENT STORE-BUS CONTROL LOGIC

1. Two buses. The 7970 compression/expansion processor allows a choice of using the main system memory alone—through the CPU bus—or in conjunction with a document store memory, which is accessed through the its store bus. Both buses are 24 bits wide.

Using advanced one- and two-dimensional compression algorithms such as those implemented in the Am7970 (Fig. 1), a 1-megabyte/page document is conservatively expected to compress down to an average value of 64-K bytes/page—a reduction ratio of more than 15:1. In many cases, the Am7970 can achieve compression ratios at least as high as 30:1. At these very high compression ratios, the net transmission time would be reduced to between 25 and 55 seconds at a 9,600-b/s communication rate.

The 7970 adheres to the T.4 and T.6 standards recommendations set forth by the International Telegraph and Telephone Consultative Committee (CCITT) Group 3 and Group 4 committees, respectively. The recommendations establish compatibility among manufacturers of facsimile equipment and have resolved many of the issues surrounding the use of compression techniques. Hence, Group 3 and Group 4 are leading the way toward the



2. One-dimensional. The 1-d compressor encodes image data into run lengths using the modified Huffman code table. Pixel runs—black or white strings of image data—use separate codes that reflect commonly found horizontal correlations in images.

standardization of compression in the office automation environment.

The CCITT compression and expansion techniques, based on the modified Huffman and modified relative element address designate (READ) codes, remove and regenerate only the image redundancies so that the methods are totally image-preserving: they are, fundamentally, statistical encoding methods. The compression ratios that can be expected when using these methods will vary according to how closely a particular image corresponds to the makeup of the images that were initially used in deriving the encoding algorithms.

In the derivation of the algorithms, the CCITT collected eight representative test documents, which encompass a wide variety of images—line drawings, foreign-language text, mixed text and graphics, for example. The CCITT then used these documents to quantify the standard compression ratios, thereby establishing an overall performance measure.

One equipment design consideration affecting compression/expansion is where to locate the compandor in the system: Does it belong locally at the controller level, in between at the input/output channel level, or globally at the main central processing unit level? This design tradeoff must be evaluated on a case-by-case basis.

However, each of the configuration options inherently requires some buffering of the data to minimize variations in compandor throughput: differences in document statistics result in instantaneous variations in compression ratios. In the 7970, the buffering is handled optionally by an on-chip, dual-bus direct memory access (DMA) controller—the 7970 supports the choice of using either the main system memory exclusively or a special local buffer store, or document store, in conjunction with the main memory. Thus the designer is free to place the 7970 in the path that best suits the overall system performance requirements.

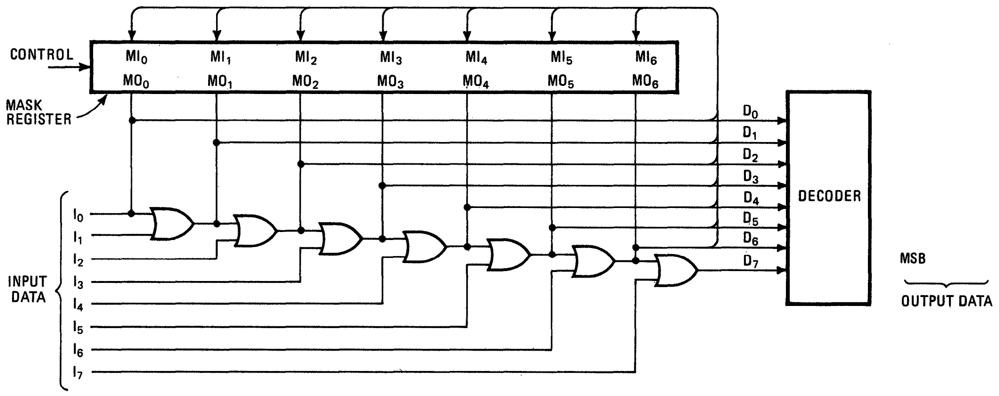
As the preceding implies, positioning of the compandor and its buffers is critical and applications-dependent, but equally important is the question of how much buffer size is needed. Because CCITT standard data compression is statistical, the exact size of the compressed data for a given image is unpredictable—it depends on how well the content of that data corresponds to the CCITT's average statistics. Thus, the temporary buffer size allocated to the code buffer for the compressed image may need to be as large as the space used in the line buffer to hold the original image—an uncertainty that has discouraged the implementation of compression in the past.

The 7970 allows total buffer flexibility by providing a 24-bit linear address range on both buses. This address range allows for up to 32 megabytes of data to be allocated—16 megabytes for the line buffer and 16 for the code buffer. On-chip registers specify the location of the line buffer in main memory, or in document memory, along with the starting address of that buffer and its assigned length. Similarly, internal registers specify the location of the code buffer in main memory or in document memory, its starting address, and its assigned length.

Compression interrupt

Equally important, the 7970 has a built-in negative-compression-detection interrupt for cases such as grey-scale drawings or highly random data in which the coded image might even expand a little. The 7970 monitors the data during compression and, in the event that the coded image is larger than the original, it interrupts the CPU to notify it to take exception, making more efficient use of the system's line and code buffers. Buffering with the 7970 therefore can be efficiently partitioned into multiple segments, which are useful in ping-ponging strategies, queuing, and full-duplex compression and expansion.

On the other side of the coin, all of this coded data must be returned to its original form of pixels, for hard copy or for display. For this, the 7970 architecture allows for expansion and compression to occur independently—the compressor from its line and code buffers,



3. Manipulating bits. The parallel encoder plays a crucial role in 1-d compression. The binary weight appearing at the output of the decoder defines a net black or white run, after the contents of the bit-length register from the previous run have been subtracted.

and the expander, simultaneously, from its own line and code buffers. Simultaneous operation of the compressor and expander is loosely termed full-duplex operation.

Another programmable feature of the 7970 is its ability to completely describe the layout of the page; the compression and expansion sides each have four registers allotted for this purpose. The paper size registers specify the length of the current scan line, or paper width, in terms of pixels. Although Group 3 equipment is limited to compression and expansion of scan lines up to 2,623 pixels, the 7970 accommodates paper widths up to 16-K pixels. This width, at a resolution of 300 pixels/in. corresponds to a document width of about 4½ ft. Thus the 7970 can easily accommodate even very large, E-size engineering drawings at this resolution.

The top, left, and right margin registers allow for the specification of the amount of white space to be encoded before the image encoding commences; this follows the Group 4 guidelines for overlaying text information with graphics information. By providing the margin control registers, the 7970 supports the capability for specifying windows of image information which can later be combined with text or other data that does not lend itself well to compression.

Programmable registers

The versatility in the 7970 offers the designer a wide variety of configurations. A total of nearly 40 different programmable registers allows for hundreds of option combinations. The compression and expansion command registers, for example, specify global operation options—the least significant bit, bit 0, of these registers is called the Go bit. The system software is intended to set, or reset, this bit in order to initiate either compression or expansion operations, respectively.

Bits 1 and 2 of the compression and expansion command registers are collectively called the operation control field—this field specifies the global context of the upcoming compression or expansion operation. One operation, reset, refers to the clearing of the internal working register contents, the process control flags, etc., by software; this command is available in addition to the external hardware reset.

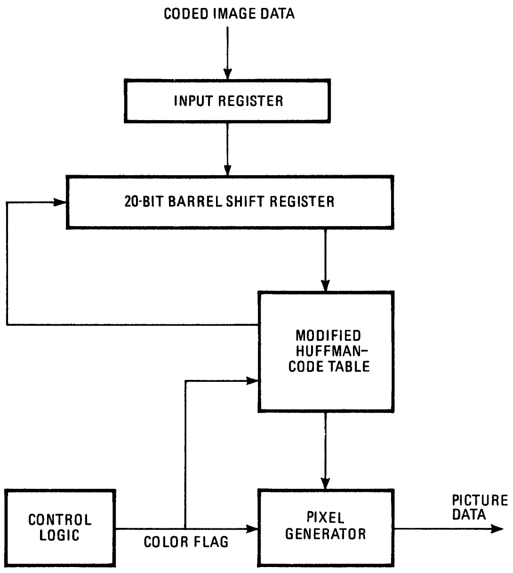
When the single line command is used, only one effective line of data from the source buffer will be compressed or expanded before termination. On the other hand, the multiline command allows the compression or expansion operation to continue until an internal word count register is exhausted.

The compression and expansion command register also reflects the interrupt status (bit 3) and source and destination control (bits 4 and 5). Bits 6 and 7 are referred to as the mode control field. This field allows the selection of three modes: one-dimensional (1-d), two-dimensional (2-d), or transparent. Group 4 specifies the transparency mode in order to efficiently handle data that will not compress. When operating in the transparent mode, the 7970 does not modify image data; data that is fed to the compandor simply passes through unaffected.

The 1-d mode specifies the use of the modified Huffman code, as recommended by the Group 3 specification. During this mode of operation, the 7970 uses the various register options that apply to the 1-d mode, ignoring all others, whereas in the 2-d mode, the modified READ code is used according to the Group 3 and Group 4 recommendations. Again, there are many register options available for specifying the 2-d parameters that will be in effect during this mode.

Despite its flexibility, the 7970 does not sacrifice performance—it is capable of a throughput of 2 to 8 Mb/s using a 5-MHz clock. Although a throughput this high may not be necessary in all applications, data compression in the emerging integrated-work-station environment does require it—in fact, compandor throughput is critical for any application in which image information is to be expanded and scan-converted for display, or for any application in which the data is to be compressed for high-performance storage on peripheral devices.

For example, a work station that is to display a document originally scanned at 300 by 300 pixels—as well as



4. Picture-perfect. In expanding an image back to the exact form it was in before being compressed, a barrel shift register is employed to position the data for a lookup in the modified Huffman table.

to transmit that document to a remote laser printer—requires not only the expansion of the coded data for transmission, but also some form of resolution conversion so that data can be displayed on a screen. Thus, even with multilevel display buffers, high compandor throughput is needed to update a display screen within a reasonable amount of time. In systems requiring very high throughputs, up to four 7970s can be implemented in parallel for a throughput in the range of 7 to 24 Mb/s. In such a system the document is partitioned into segments, each of which is processed separately by a 7970. The only limiting factor in a parallel system is that the DMA bus arbitration latencies will accumulate and become self-limiting.

CCITT algorithms

The 7970 is based on a unique implementation of the CCITT standard compression and expansion algorithms. A patented approach to the processing of compandor data, this invention is a high-speed, run-length encoding/decoding method that employs the parallelism usually found in the grouping of pixels which make up two-tone image data. The innovation is an architectural extension of the same concepts that are used in the compression and expansion of the image data itself: clock generation, as well as pixel group encoding/decoding, is a function of run length. With these techniques, the encoding and decoding operations can be performed more efficiently by clocking run blocks only on color changes.

Group 3 recommends a 1-d data compression method, the modified Huffman code, which takes advantage of the horizontal correlations that are commonly found among pixels within a two-tone image. The 1-d method efficiently encodes an image into run lengths using separate code tables for black and white runs in accordance with their separate statistical frequencies of occurrence.

In general, a white pixel run consists of consecutive 0s, and a black pixel run consists of consecutive 1s. The code tables, which contain terminating codes for run lengths of between 1 and 63 pixels—with 1 bit per pixel—are based on the run-length redundancy-reduction techniques. Additionally, sets of makeup codes are used to encode multiples of 64 pixels—up to 2,623 pixels. The 7970 optionally supports multiple makeup codes up to 16-K pixels. Each line terminates with an end-of-line code word, which is 11 or more zeros followed by a 1.

The 1-d compressor (Fig. 2) of the 7970 is compatible with Group 3. In a simplified illustration of the encoding scheme, the 7970 is positioned for the start of a new line of image data. At this point, the input register of the 7970 has received 8 bits of data, say 10001111 (scanned from the least significant to the most significant bit), which represents one black pixel followed by three white pixels followed by four black pixels.

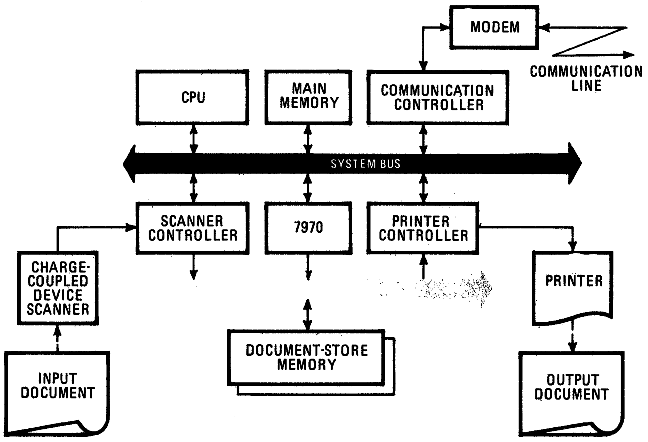
The data is impressed on the parallel encoder (Fig. 3) inputs, I_0 through I_7. Initially, the contents of the parallel encoder are equal to the contents of the bit-length register so that a white run length of 0 is first obtained in the run-length counter, and the color flag is set to 0 in keeping with the CCITT convention that a new line always begins with a white run, even if that run amounts to 0. Thus, initially, the decoder inputs D_0 through D_7 of the parallel encoder are all 1s, which decode to all 0s at the four outputs.

Color flag

Since by definition the next run will be black, the color flag is toggled to 1, and the input data I_0 through I_7 is exclusive-ORed with this flag to invert the input data to the value 01110000, which corresponds to 01111111 at the decoder's input and 1000 (LSB to MSB) at its output. Therefore, a black run length is obtained by subtracting the previous contents of the bit-length register, 0, from the current contents of the parallel encoder, 1, for a net black run of 1 - 0 = 1. Again, the color flag signal is toggled to 0 for the upcoming white run, and the data I_0 through I_7 reverts to 10001111. However, the first run is no longer needed now, so the mask register contains a 0 in mask output bit 0 (MOO) such that the decoder inputs D_0 through D_7 are masked to 00001111. The output of the decoder of the parallel encoder is now 0010, or 4. This value is subtracted from the contents of the bit-length register, one, and are loaded into the run-length counter as a 4 - 1 = 3 pixel run length.

The process continues for the remaining black run, which is a special boundary condition—after masking operations on the preceding runs, the decoder of the parallel encoder contains a binary weight of 0001, or 8, at its output. Because the next byte of image data might

5. Facsimile fever. The 7970 handles both Group 3 and Group 4 facsimile applications; thus, the chip can be used in general work station configurations. However, scanners and printers might also interface directly to the document store bus.



contain the continuation of this current black run of 8 - 4 = 4, it is necessary to retain this run for possible combination with the run data from the next byte.

The 1-d expansion architecture (Fig. 4) is the counterpart to the 1-d compression structure, which replaces the redundancies within a line originally removed during the compression process. However, most images exhibit very strong vertical correlations from one line to the next, in addition to the horizontal correlations, which makes two-dimensional (2-d) coding techniques desirable. In 2-d compression, the pixel run can be specified by its similarity to the position of a run on the previous line.

Both Group 3 and Group 4 outline a 2-d data compression standard, the modified READ code. This one is a line-by-line coding method in which the position of each changing pixel on the current or coding line is encoded or decoded with respect to the position of the corresponding reference line—usually on the line directly above. A changing element is defined as any pixel whose color—black or white—is different from the color of the previous pixel along the same scan line. After the current line has been coded, it becomes the reference line for the next coding line, and so forth.

Operating modes

The 7970 is fully compatible with the Group 3 and Group 4 recommendations for 2-d encoding, the fundamentals of which include three modes of operation:

■ Vertical mode, which is identified when the coded lines' changing element's displacement from the reference lines' changing element is less than four pixels to the right or left. This relative distance can then take on one of seven values such as vertical right 1 pixel [VR(1)] or vertical right 2 pixels [VR(2)]. A unique code word represents each of these differentials.

■ Horizontal mode, which is used when the pixel displacements are four or more in either direction. The run lengths are then encoded using the modified Huffman techniques from the 1-d coding scheme.

■ Pass mode, which is an exception condition and is identified when the black or white runs on the reference line are not adjacent to the corresponding black or white runs on the coding line, and therefore will not lend themselves to compression. In this case, the coding line is realigned with respect to the reference line and he process of evaluating the pixel differentials for coding in either the vertical or the horizontal mode is reinitiated.

The implementation of the 2-d algorithm is more complicated than that of the 1-d structure. However, if the 2-d algorithm is broken down into subsections, similarities to the 1-d architecture can be noted. For example, the front-end of the 2-d compressor is simply two sets of the same front-end logic as is used in the 1-d compressor: one set for the reference line and one for the coding line. This logic consists of input registers, the reference input register and the coding input register, some exclusive-OR logic which is toggled by a color flag, and parallel encoders.

The 2-d compressor, however, requires an adder for computing the position of each changing pixel on the reference line as well as its counterpart adder for the coding line. In addition, a third adder is required to take the differential value or offset between these positions. Also needed is a multiplexer for the reference line, which is used to check for exception conditions. Other components that are required in the 2-d compressor that are not required for 1-d compression include two 2-d complementers, several address registers, and the modified READ code table.

The 2-d expander logic is simply the compressor counterpart, derived very similarly to that of the 1-d expander except that the 2-d expander must operate on two lines—a reference line and a coded line. Given the suitability of the 7970 for both 1-d and 2-d applications, this chip can be used in many different system configurations—Fig. 5 shows how the 7970 might be used in a typical Group 3 or Group 4 facsimile system.

In such applications, the scanner and perhaps a laser printer would also reside on the document store bus. However, this might require a bus arbiter and perhaps the transceiver buffering of the 7970 because the document store bus does not have arbitration control logic. But in many systems, scanners and laser printers do not need to operate independently of the system's CPU—hence the 7970's ability to release its document store bus control through software is not only adequate, but desirable. The main system bus would then be used as the master controller for the incoming or outgoing facsimile transmissions, as well as for the ongoing scanning and printing operations. □

Data Security with Fast, VLSI Circuits

Juergen Stelbrink

Advanced Micro Devices, Sunnyvale, California, U.S.A.

History

Cryptography is almost as old as civilization, developing from the human desire for privacy. For example, a simple substitution cipher (letters are replaced by other letters) invented by Julius Caesar protected confidential messages in the gallic wars. Other crytographic algorithms scrambled the position of letters within certain text (transposition cipher). Over the past several centuries, many military victories depended on breaking cryptographic algorithms.

Summary

Computer crime, consisting of unauthorized data manipulation and theft, causes estimated annual losses in the range of $1-3 billion. Since many such incidents are not being reported or even detected, the real figures might be much higher. Computer networks communicating over the standard telephone network almost invite intruders equipped with simple computing resources to gain unauthorized access to masses of sensitive data. Consequently, data security will become a keyword in upgrading existing computer networks or designing new ones.

Standardization

In January 1977, the National Bureau of Standards (NBS), a U.S. government agency, standardized a cryptographic algorithm known as the Data Encryption Standard (DES) /1,2/. Employing the same key for encryption and decryption, it is an implementation of the private key system. $72*10^{15}$ encoding possibilities given by the 56-bit key make it virtually impossible to retrieve the original data without knowing the key. The ciphered text is a complex, non-linear function of the input text and key, involving XORing, substitutions, block swapping, and key subset selection. These binary operations are well-suited for hardware implementation and allow higher data speed and better data security than software ciphering.

The National Bureau of Standards has defined three implementations of the DES algorithm, each being optimized for certain applications:

ECB Electronic Code Book (fig 1) is the direct implementation of the DES algorithm. The analogy to a code book arises since

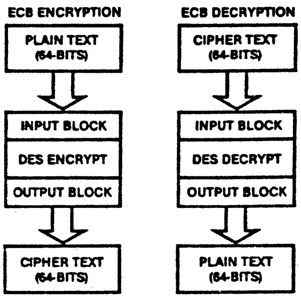


Figure 1. Electronic Codebook (ECB) Mode

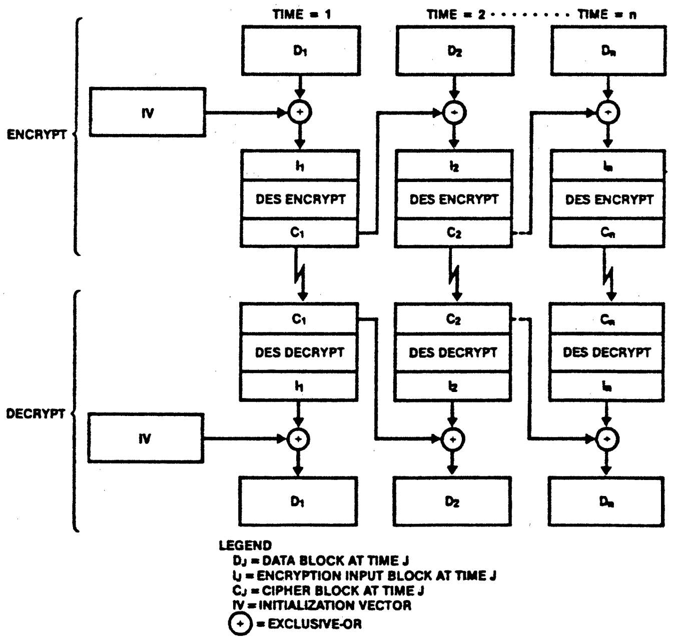

TIME = 1
TIME = 2 ········ TIME = n
ENCRYPT
DECRYPT
IV
D1
D2
Dn
I1
I2
In
DES ENCRYPT
DES DECRYPT
C1
C2
Cn
LEGEND
DJ = DATA BLOCK AT TIME J
IJ = ENCRYPTION INPUT BLOCK AT TIME J
CJ = CIPHER BLOCK AT TIME J
IV = INITIALIZATION VECTOR
⊕ = EXCLUSIVE-OR

Figure 2. Cipher Block Chaining (CBC) Mode

the same plain text always generates the same ciphered text for a given cryptographic key. Data is handled in blocks of 64 bits each. Since these data blocks are independently ciphered, this mode is suited for data storage.

CBC Chain Block Cipher (fig 2) overcomes the weakness of the ECB mode (identical blocks of plain text always generate identical blocks of ciphered text). Since in the CBC mode the current block affects ciphering of the next block, a sequence of identical input blocks will create a sequence of changing output blocks. This involves an error extention characteristic, protecting the data sequence against fraudulent data insertion or deletion. These features make CBC best suited for high-speed data communications. Some hardware implementations can even handle Ethernet data rates on the fly, eliminating the need for buffering.

CFB Eight-bit Cipher Feedback (fig 3) operates similar to CBC but is optimized for medium-speed, character-based data communication, since it is based on 8-bit blocks (bytes) rather than 64-bit blocks.

Data Communication

The DES algorithm perfectly meets the requirements of point-to-point data communication via satellite, between a host computer and its remote terminals, and in star networks. Both sender and receiver use the same key for encryption and decryption. Changing these keys frequently improves the system security. A two-level key system consisting of a master key generating session keys allows secure distribution of new session keys via the communication link.

One form of data communication, electronic mail, is beginning to replace paper mail. Obviously, the new mail system should maintain or improve upon the addressability and authenticity of the paper mail system. Authenticity ensures that the message has not been tampered with and identifies the sender by analyzing his signature. Addressability and authenticity are features that a single-key system does not support directly. A dual-key system such as the public key system first introduced by Diffie and Hellman in 1976 /3/ overcomes these shortcomings by offering separate keys for encryption and decryption.

Each network subscriber generates a pair of keys; one key he keeps for himself as a "private key," and the other he enters into a public key file. Addressability is achieved by exchanging transmitted messages in the following manner: The sender takes the key (address) of the subscriber, who in turn receives the message, and encrypts it. Since the receiver is the only one in the network with the matching decryption key, only he can interpret the transmitted message.

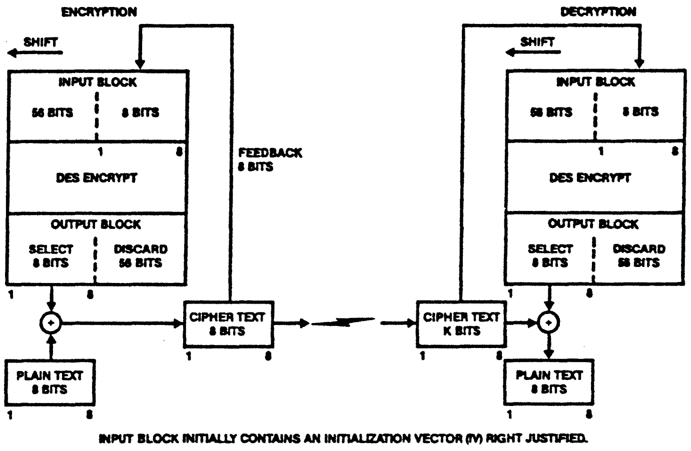


Figure 3. 8-Bit Cipher Feedback (CFB) Mode

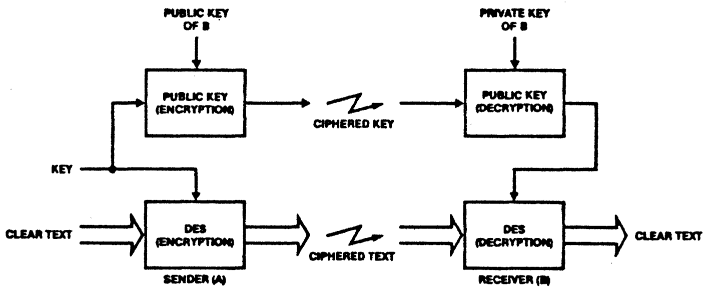


Figure 4. Hybrid System

Authenticity is ensured if the sender encrypts the message first with his private key. This allows any network subscriber to verify the identity of the originator. By encrypting twice, first with the private key of the sender and then with the public key of the receiver, this system ensures both addressability and authenticity.

In both single-key and dual-key systems, the length of the key determines the security level of the cryptosystem (the longer the key, the greater the security). For DES this is fixed at 56 bits, whereas the dual-key system allows a flexible key length. To compete with the code-breaking capabilities of today's supercomputers, the key must have between 100 to 200 digits.

However, dual-key (public key) systems are slow since they involve multiple-precision arithmetic on very large numbers (>100 digits). The functional advantages of a dual-key system (electronic mail, more security) can, however, be combined with the advantages of single-key cryptosystems (speed and availability of dedicated VLSI circuits) to form a hybrid system (fig 4).

Silicon Implementation

Advanced Micro Devices produces a family of Data Ciphering Processors (Am9518, AmZ8068, and Am9568) /4/ (fig 5 and 6) supporting the single-key DES cryptosystem. All three devices feature very high data throughput (up to 14.3 Mbit/s), and a bus interface which is compatible with most microprocessors.

Key Registers

The Data Ciphering Processor (DCP) has three separate write-only key registers holding the Encryption key; Decryption key, and Master Key. Separate encryption and decryption key registers eliminate the time-consuming task of reloading keys when changing from encryption to decryption or vice-versa.

To enhance the security of data communication networks, it is desirable to use different keys for each session. Therefore, the system must support a simple and secure method of key distribution. In DCP-based systems, this feature can be implemented via the Master Key register. Before starting data transmission, a session key is randomly generated. This session key is encrypted using the Master key and securely transmitted over the communication link. The DCP in the receiver station takes the encrypted key, decrypts it, and puts it directly into the appropriate register. Because the clear encryption or decryption key is never seen outside the chip, and because all key registers are only writable, not even a system programmer can jeopardize the security of the system by accessing the clear key. To ensure correct key entry, each byte loaded is checked for parity.

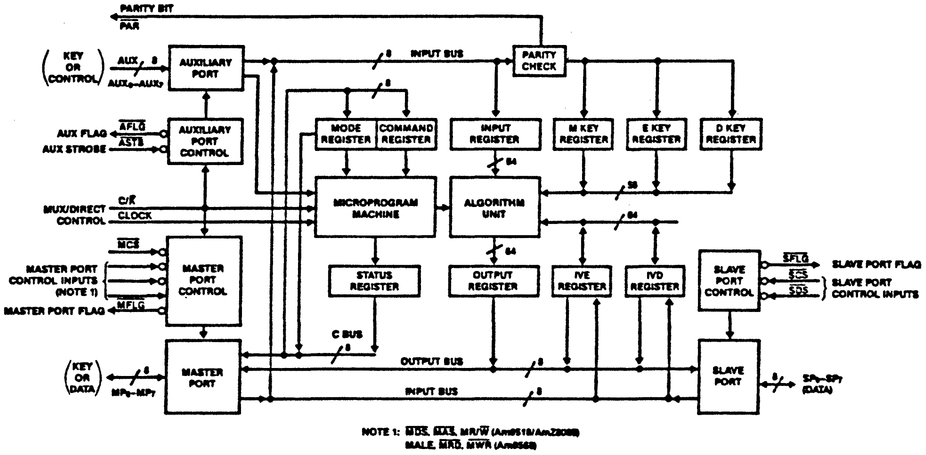


Figure 5. DCP Block Diagram

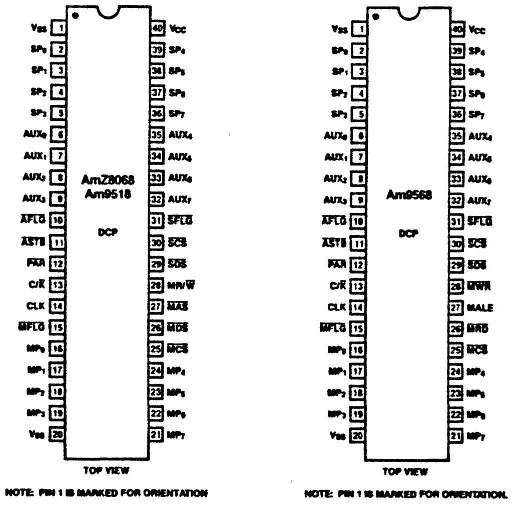


Figure 6. Connection Diagrams

Very High Data Throughput

With a data ciphering throughput of up to 14.3 Mbit/s, the DCP easily satisfies the speed requirements of most applications. For instance, it allows data ciphering on the fly as data is transferred between the system and disk drives (typ. 5 Mbit/s) or as data is transferred between the system and an Ethernet controller (10Mbit/s). These on-the-fly transfers eliminate the need for temporary buffering, improving system performance and reducing system cost.

Applications

All three devices can be used in a large variety of systems, including dedicated controllers, communication concentrators, terminals, modems, and peripheral task processors. Typically, the DCP is interfaced directly to standard microprocessors. For this environment, the DCP supports an 8-bit peripheral interface with a multiplexed address data bus (Master Port). Commands, encryption and decryption keys, and data are passed through this port (fig 7). The Master Key is entered via a separate Auxiliary Port. This enhances system security by separating the paths for keys and data.

For high-performance systems, single-port operation becomes a bottleneck reducing data throughput. The DCP solves this problem with a third 8-bit port, the Slave Port. In the dual-bus configuration (fig 8), input data can be loaded via one port into the DCP while the processed data is removed in parallel from the other port. Figure 10 shows the data flow for this pipelined, dual-port operation.

Dedicated disk or communication controllers based on microprogrammed logic can operate the DCP in Direct Control Mode (fig 9), in which the DCP is controlled by external pins rather than internal registers.

References:

/1/ U.S Department Of Commerce/ National Bureau of Standards "Data Encryption Standard", FIPS PUB 46, Jan 1977

/2/ U.S Department of Commerce/ National Bureau of Standards "DES Modes of Operation", Sept 1980

/3/ Diffie, W. and Hellman, M. "New Directions in Cryptography" IEEE Transactions on Information Theory, IT-22(6), Nov 1976

/4/ Advanced Micro Devices "Data Ciphering Processors, Technical Manual", 04862A, April 1984

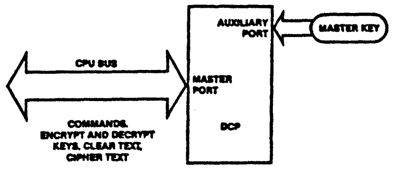


Figure 7. Data Flow for Single Port Configuration, Multiplexed Control Mode

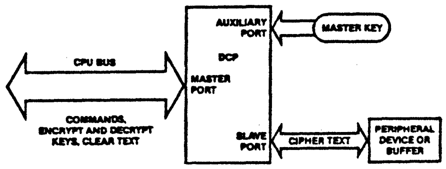

AUXILIARY PORT
MASTER KEY
DCP
CPU BUS
MASTER PORT
COMMANDS, ENCRYPT AND DECRYPT KEYS, CLEAR TEXT
SLAVE PORT
CIPHER TEXT
PERIPHERAL DEVICE OR BUFFER

Figure 8. Data Flow for Dual Port Configuration, Multiplexed Control Mode

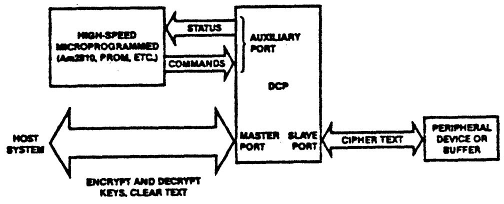

HIGH-SPEED MICROPROGRAMMED (Am2910, PROM, ETC.)
STATUS
COMMANDS
AUXILIARY PORT
DCP
HOST SYSTEM
MASTER PORT
SLAVE PORT
CIPHER TEXT
PERIPHERAL DEVICE OR BUFFER
ENCRYPT AND DECRYPT KEYS, CLEAR TEXT

Figure 9. Data Flow for Dual Port Configuration, Direct Control Mode

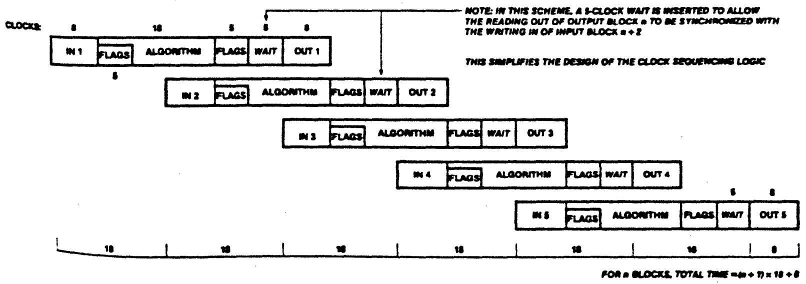

NOTE: IN THIS SCHEME, A 5-CLOCK WAIT IS INSERTED TO ALLOW THE READING OUT OF OUTPUT BLOCK n TO BE SYNCHRONIZED WITH THE WRITING IN OF INPUT BLOCK n + 2
THIS SIMPLIFIES THE DESIGN OF THE CLOCK SEQUENCING LOGIC
CLOCKS:
8
16
5
5
8
IN 1
FLAGS
ALGORITHM
FLAGS
WAIT
OUT 1
5
IN 2
IN 3
IN 4
IN 5
OUT 2
OUT 3
OUT 4
OUT 5
16
8
FOR n BLOCKS, TOTAL TIME = (n + 1) x 16 + 8

Figure 10. Pipeline Scheme for Dual Port Operation

Protocols and network-control chips: a symbiotic relationship

As local-network data rates head towards 100 Mb/s, interaction between protocol design and VLSI implementation determines system efficiency

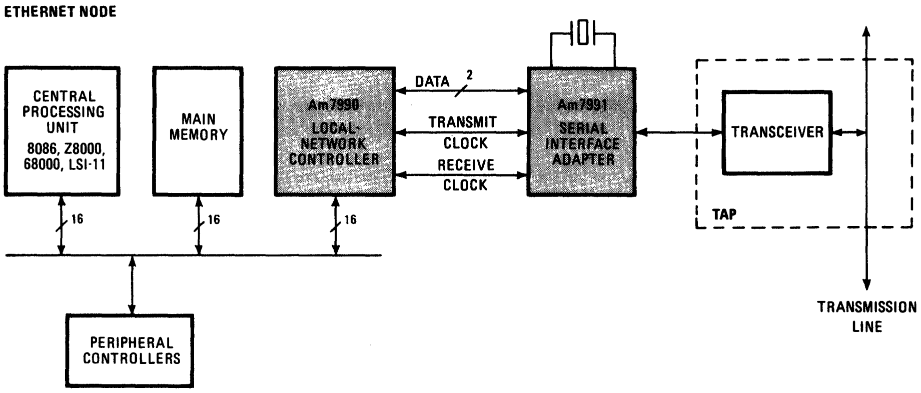
by Sunil Joshi and Venkatraman Iyer, *Advanced Micro Devices, Sunnyvale, Calif.*

□ Today's great felt need for practical local networks is yet another effect of the low cost of computing power and the resulting proliferation of computing devices. For that proliferation has in turn created very large amounts of data that must be processed and shared. How well can the devices in a local network share it, and how quickly? These questions make protocol efficiency at high data rates an important technological problem of the 1980s. To solve it, more attention must be paid to the tradeoffs involved in local-network packet design, access schemes, encoding, and duplexing, and also to the ways in which local-network bottlenecks emerge.

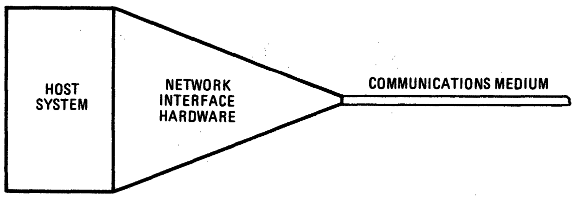
A chip set, tentatively called Supernet, that deals with these problems is now in the early stages of development. Much as two Ethernet chips (Fig. 1)—the local-network controller [*Electronics*, Oct. 6, 1982, p. 92] and the serial-interface adapter [*Electronics*, Nov. 17, 1983, p. 148]—connect that 10-megabit-per-second network to data-generating and receiving devices, the new set will link local networks at speeds up to 100 megabits/s. But the difficulties are formidable indeed. Besides lowering the cost of devices, very large-scale integration has raised computer-processing speeds by orders of magnitude. Kilobit-per-second network communications, which were adequate a few years ago, just will not do for present-generation products, which communicate in the megabit-per-second range.

Some manufacturers of computer systems may very well be tempted to keep pace with these ever-increasing speeds by setting out on a potentially dangerous course: increasing the speed of communications protocols without taking into account all aspects of the relationship between the design of protocols and VLSI implementation. As local-network speeds rise to 50 Mb/s and beyond, however, these factors can no longer be ignored. In fact, tradeoffs in local-network efficiency, response time, performance, and cost can be had at all seven layers of the International Standards Organization's reference model for computer communications: the open-systems interconnection model.

Computer networks are usually classified as front- or back-end systems. A front-end network connects such



1. A standby. The high-speed chips of the future will require special high-speed protocols, unlike these two Ethernet chips—the serial interface adapter and its local-network controller—which connect that 10-Mb/s network to data-generating and receiving devices.



2. A funnel. The interface between a network communications medium and a host computer acts like a funnel. The computer feeds data to the wide end at 8, 16, or 32 bits, while the network interface converts the flow into a serial stream of bits at the apex. This funnel is bidirectional.

devices as terminals, word-processors, printers, and personal computers to bigger computers. A back-end network connects storage devices, such as disk subsystems, to other devices, such as laser printers.

Front-end networks can run at lower speeds than back-end networks, which must operate at very high speeds because of the large amounts of data they transmit. Protocols like Ethernet are oriented to the front end; schemes like the 50-Mb/s Hyperchannel are suited to back-end networks.

Changing classifications

As technology changes and the need for better communications services grows, the traditional classification of front- and back-end networks is changing. For example, the bandwidth requirements of such services as teleconferencing and mixed-mode electronic mail (including voice and text) can exceed 50 Mb/s. Front-end networks that provide such services must have bandwidths similar to those of back-end networks. Of course, as fiber-optic technology becomes cheaper, its hundreds-of-megabits-per-second capabilities will be exploitable both in front- and back-end applications.

The transition to higher speeds and better services makes it tempting to save money by just raising network data rates and hanging on to compatibility with existing protocols. There are pitfalls, however. It is not, for example, cost-effective to design all network hardware to run five times as quickly as, say, Ethernet if the network data rate goes up by a factor of five. In fact, modifications to the network protocol can produce many improvements in performance—without a corresponding increase in the cost of hardware. The main question is how this feat will be accomplished for a local network's often-used serial-communications lines.

Serial-data transmission dispenses with costly parallel-line data links and connects devices by running just one cable between them, so that it is quite popular for interfacing data-generating and receiving equipment separated by more than a few feet. Since the technique transmits data bits serially, the transmission line must provide a high bit rate for reasonable throughput.

An inexpensive transmission line is desirable, too, however. Twisted pairs are rather cheap, but they can be noise-sensitive if not shielded. Coaxial cable makes higher data rates possible but costs more than twisted pairs do. Fiber-optic cable can launch data-transmission rates into the multihundred-megabit range but is now expensive.

Network-interface hardware mainly takes data back and forth between a host computer and a local network's transmission line. In the network interface, certain functions, such as virtual circuit connections and error recovery, can be performed in either hardware or software. Software is typically slower than dedicated hardware, which is therefore the better way—especially VLSI chips.

The protocol bears directly upon the hardware implementation. In fact, a well-defined protocol can simplify an implementation without forcing the network to pay any performance penalty.

A funnel

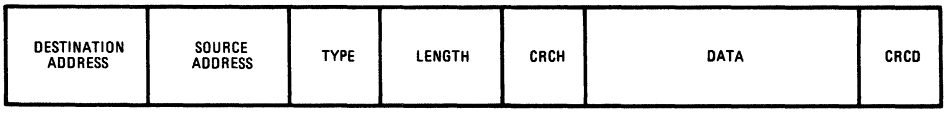
A network interface between a host computer and a transmission line is like a funnel (Fig. 2). At the funnel's mouth, the host microcomputer, minicomputer, or mainframe feeds data on an 8-, 16-, or 32-bit-wide interface, respectively. On the thin end of the funnel, a bit-serial interface to the network transmission line performs parallel-to-serial conversion on the computer's data while transmitting it. The bidirectional funnel performs serial-to-parallel conversion when receiving.

Should the data-handling rates of the host and the network be mismatched, or should the two operate asynchronously, the network interface may have to provide data buffering. Apart from serializing the parallel data and buffering it, the network interface can also manipulate, process, and reorder the data.

While data is transmitted, the network-interface funnel follows the "venturi" principle. In this analogy between the data stream and fluid in a funnel, data moves much faster in the funnel's narrow end than at its mouth. In a 100-Mb/s network, for example, a 16-bit-wide processor will have to supply data at a rate of about 6.7 megawords per second, a 32-bit wide processor at only 3.4 megawords/s. These speeds, within the range of today's bit-slice processors, are not far from the range of some conventional MOS microprocessors.

The network-interface hardware's processing of the data stream can be regarded as a kind of perturbation that causes turbulence inside the funnel. If data is operated on in parallel while it is 16 or 32 bits wide, the turbulence is not great. If the operations take place close to the transmission line—the narrow end of the funnel—faster hardware is required because the data bits pass by much more quickly.

A high-speed network at 100 Mb/s would certainly require emitter-coupled-logic processing technology very close to the transmission line. If the data can be converted to 8- or 16-bit parallel words, TTL or MOS devices can do all of the processing. Should a protocol require a quick response, storage, or processing on a bit boundary, high-speed—to say nothing of high-cost—hardware is essential. For the sake of economy, system designers must



3. Protection. In a typical packet format, the fields before the data are called the packet header. The header and the data in this packet are protected by two separate cyclic-redundancy-check fields—one to protect the header (CRCH), the other to protect the data field (CRCD).

define the functions of protocols in a way that permits the use of dense integration technologies, such as MOS or complementary-MOS.

In the not-so-distant past, network data-transmission rates were very low. Transmission bandwidth was therefore at a premium, and heroic efforts were made to economize on it. Packet-header lengths, for example, were kept to a minimum, and different packet fields were highly encoded and organized on bit boundaries. All processing thus had to be done at the bit level, at the tip of the data-communications funnel. This was possible, however, since data-transfer rates were low.

Newer protocols must provide for changing technologies. As 8-bit processors lose market share, for example, future protocols should be optimized for 16- and 32-bit microprocessors. As fiber-optic technology becomes available, appropriate user services should be provided by streamlined protocols, not imposed by the available bandwidth as they are at present.

Move that bottleneck

Larger bandwidths have shifted the local-network bottleneck from transmission to processing. Newer protocols try to make processing as simple as possible and to remove the packet format's bit edges—in other words, these protocols present data in a way that makes it unnecessary for the processor to worry about each bit. High-speed protocols, though typically organized on byte boundaries, could be organized on word (16-bit) or long-word (32-bit) boundaries as well. Long-word boundaries match the packet format to the processor's width and make processing easier and faster.

If processing does become a bottleneck, the buffer memory must be enlarged to accommodate the queued-up packets. Frequent overflows invoke flow-control mechanisms and also create processing overhead, and a large buffer memory adds to costs. It is therefore really essential to avoid processing bottlenecks.

The packet size of current protocols is the result of tradeoffs. In a packet-switched (as opposed to circuit-khitched) network, packets should be kept to a few to prevent "network hogging" by the transmitter. When packets are small, every message is chopped up into several packets, and since there is an overhead in the header with each packet, the network's efficiency is so much the less. Even worse, more network packets can easily increase the level of processing and create a still more serious network bottleneck.

Long packets need larger buffers, so the size of the buffer memory, too, is affected by the size of the packets. Since high-density memories are getting less expensive, however, this is less important than the processing bottleneck. The packet's size is also a function of the transmission line's bit-error rate. High error rates, which raise the possibility that long packets might be corrupted, should therefore be avoided.

In addition to data packets, several small packets contain nothing but control information. Sometimes, these packets should be padded with dummy bytes to make them longer—a procedure that may well be useful for a collision-detection protocol that must be able to detect all collisions. Padding can also reduce the constraints on hardware that may have to allow for several short packets in quick succession.

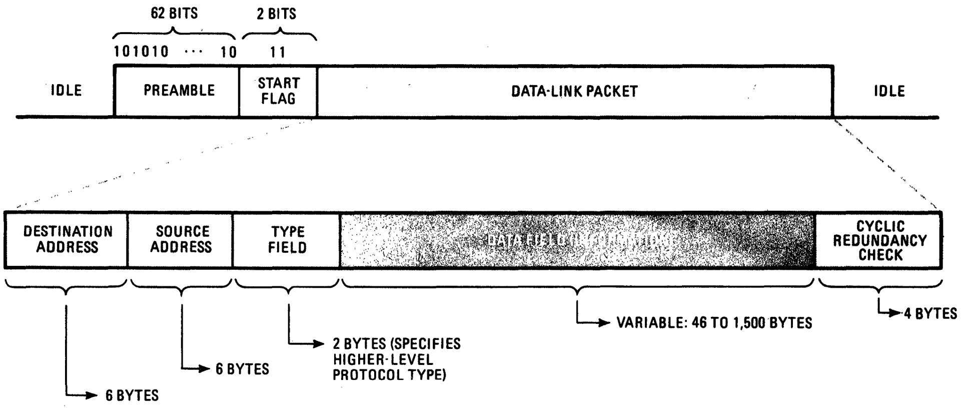
On a low-speed network, even a packet that is 1 byte long may not be at all hard to handle; because of the low network bit rate, it might well provide a reasonable time on an absolute scale. But if the packet stays 1 byte long at a data rate 10 times the former one, for example, the packet's time duration will be one tenth of the data rate, and the packet may therefore have to be processed by hardware that is 10 times faster than the low-speed network hardware.

In a typical packet format, the fields before the data (or information) fields make up the packet header. To increase the network's efficiency, conventional protocols try to keep headers short. Fields are thus highly encoded, and much software overhead is needed to determine a field's meaning. To save on wasted bits, moreover, functionally unrelated fields have been grouped together. Extended fields are used when the next data byte's interpretation depends on a few bits in the previous byte. The right approach is to tolerate longer headers with functionally meaningful fields that fall on byte or word boundaries, so the central processing unit can interpret them quickly.

Addressing fields

A local network's destination address can usually be either a multicast address or the address of a physical device. High-speed networks for back-end systems typically consist of a few nodes in a small area, so the address fields are rarely more than a byte or two.

Multicast addressing is short and simple as well, to prevent the processing and the storage of packets that do not belong on the node. In fact, the small number of nodes and the high bandwidth of networks that do not use multicast addresses make it quite possible to replace



4. The old. Ethernet's physical and data-link-layer packet formats have been standardized by the IEEE 802.3 committee on local-network standards for 10-megabit-per-second data rates. Higher data rates require new protocols and different formats.

a multicast packet with several packets that are physically addressed.

Quite similar considerations of addressing also apply to the source address—which must, however, be physical. A multicast source address just does not have any meaning, because multicasting has one source and more than one destination, but the source address is physical. Unlike the multicast destination address, it does not have to be broken down from a symbolic address into multiple physical addresses. The first bit of the address denotes the physical or the multicast mode. This design does not, however, violate the sanctity of the byte fields, because the denotation of the physical or the multicast modes merely has the effect of dividing the address space in half. Address look-up can still be performed upon the entire byte.

Front-end networks must sometimes use longer address fields. Ethernet for example, uses a 48-bit field to give every node in its universe a unique address. This procedure may make it harder to manage the allocation of addresses.

Many fields

A "length field" is almost mandatory in a high-speed bus protocol, since carrier falloff, which denotes end-of-packet in low-speed protocols, cannot be used at high speeds. As a local-network rule, any OSI layer's length field indicates the packet length in the layer above. For example, the length field at the data-link layer indicates the length (in bytes) of the network-layer packet, the data field.

Some designers like to use the length field to determine, on the fly, whether or not sufficient buffer space is available for packet storage. This procedure should not be used, however, in order to avoid putting any real-time constraints on the processor's ability to respond to the incoming packets. Processing can be further simplified by having all the packets fall into just a few length categories. Such fixed lengths for all data packets can simplify the allocation of buffers, among other things.

"Type field" is a generic term for fields that provide information about the packet itself, including the type, control, and access fields. They indicate the contents of the packet, the action that must be taken with it, whether or not an acknowledgment is needed, and where the data can be routed.

The header's cyclic-redundancy-check field, characteristic of high-speed protocols, protects the header, which is usually long. A packet with a bit error in the destination-address field can therefore be rejected at an early stage, avoiding needless processing. The CRC field also ensures that in the event of an error in the length field, buffers do not get filled with garbage bytes.

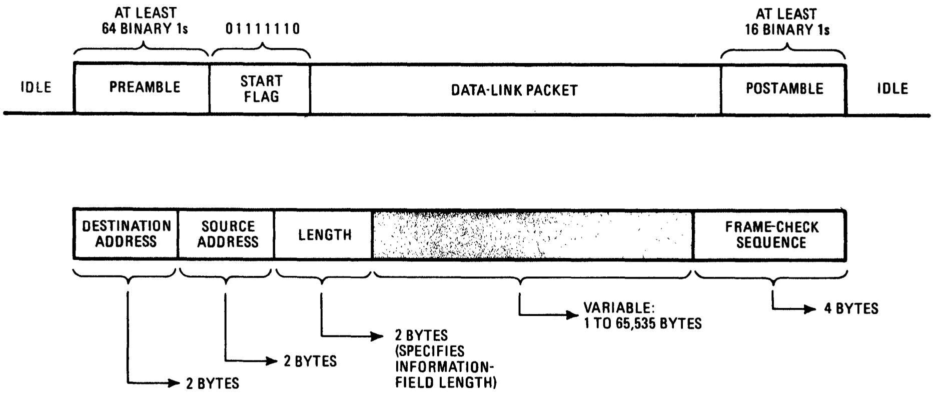
Two in one

With two CRC fields used in one packet—one protecting the header, the other the data (Fig. 3)—longer packets can be transmitted than would be possible with one such field. High-speed protocols lean to a longer CRC polynomial, 32 bits, than do low-speed protocols, which may use 16-bit polynomials. The Autodin II 32-bit CRC polynomial, a Federal standard, is preferred for most current high-speed protocols.

In a typical packet, the data CRC protects the packet's data field, while the header redundancy check protects the header. When both are used, it makes sense to apply the same polynomial to both. The CRC computation on the data stream ought to start and end on byte boundaries, so the implementation of hardware can promote 8-bit parallel CRC generation and checking in the slower sections of the network-interface funnel.

Sometimes, ring protocols tend to compute the CRC on nonbyte boundaries because some bits in circulating tokens or flags, flipped on the fly, cannot be included in it. The better way is to define these fields so as to put all the information that must be protected by the CRC on byte boundaries.

High-speed protocols favor retransmission of packets,



5. The new. For data rates at 50 megabits a second, some variation on the American National Standards Institute X3T9.5 proposal (the logical distributed data interface) is more suitable than Ethernet's frame format.

not the correction of errors. This favoritism reflects the fact that transmission costs less than processing does.

Collision detection and other access schemes require a minimum packet length, which must be longer than the network round-trip time. At high speeds, such a design requires an undesirably long minimum-length packet. Upgrading to higher speeds also means changing the requirements of minimum-length packets and may require extensive changes in network software. Collision protocols are therefore not suited to very high-speed networks unless distances are short enough to cut round-trip delays.

To control access to the network, some high-speed bus protocols use a modified scheme for time-division multiplexing. Unused access opportunities may make it somewhat wasteful of bus bandwidth, but that is of little interest at high speeds. Token-passing rings work very well at such speeds and can also for the most part be implemented digitally—and therefore more easily than collision-detection access schemes, which need more analog circuitry.

Please confirm

An acknowledgment packet informs any node sending an information packet that it has in fact been received. The acknowledgment can include status information—for instance, whether or not the packet had CRC errors or was out of sequence. The seven-layer ISO model allows acknowledgments to be sent at the data-link, the network, and the transport layers, and at other layers as well. In fact, acknowledgments are sometimes sent on more than one layer.

Certain protocols require immediate acknowledgment at the data-link layer, where the response has to be sent in a short window after the packet arrives. This immediate response imposes certain hardware restraints because a minimal amount of processing must generate the response packet. If the response window is a large one, the received packet can be processed at the wider section of the funnel, where processing is more simple. A quick response means that the packet will not have enough time to trickle back to the low-speed logic; more high-speed hardware will be necessary.

Immediate acknowledgment not only complicates hardware but also, in many cases, does not even serve any useful purpose, since the node that gets the acknowledgment takes no immediate action on it. The node needs time to interpret the packet, which may be waiting in the receive queue. An immediate acknowledgment may be useful in a very noisy environment, where faulty packets are common and frequent retransmission is needed. But well-shielded or fiber cables prevent such abnormal conditions from occurring often.

Should some kind of acknowledgment be needed at the data-link layer, acknowledgment for a group of packets is better than acknowledgment for a single one; the sending node can dispatch a predetermined number of packets without expecting any acknowledgment at all. Since no routing occurs in local networks, acknowledgments cause no problem with packet sequences, which stay the same. If a packet comes in with an error, processing may be easier if the whole group of packets is sent again. Logical or multicast addressing makes immediate acknowledgments meaningless because several nodes cannot use the same time slot for them.

Packet streaming

Some applications require protocol features that would certainly not otherwise be desirable. One such feature, packet streaming, is used for disk transfers. Once the network transmission line has been acquired by a node, the node keeps control of it until a complete track or so on the disk has been transmitted. Each sector is a separate packet. This procedure, although analogous to network hogging, is quite useful for disks because it is not appropriate to wait for a complete revolution of the disk to load each packet. Despite the possible bottleneck of hogging, packet streaming actually does raise the effec-

Parameter	Ethernet	ANSI X3T9.5
Speed	10 Mb/s	50 Mb/s
Access method	Carrier-sense multiple access with collision detection	Time-division multiplexing
Delimiter	Carrier falloff	Length of field
Address space	6 bytes	2 bytes
Transmission method	Baseband with Manchester encoding	Broadband with clock and data on separate channels
Maximum packet size	1.5-K bytes	64-K bytes
Type field	Yes	No
Multicast addressing	Yes	No
Packet streaming	No	Yes

tive throughput of each node in a disk system.

Packet streaming occurs in a protocol optimized for a specific application. Protocols that incorporate this procedure may not work well in offices with voice and video transmissions, because they increase time latencies for the transmission of ordinary packets.

Encoding schemes should usually be simple. The encoding hardware is always at the high-speed section of the funnel, where complexity can add to costs. Encoding schemes can also increase the transmission line's bandwidth utilization many times over. Manchester encoding, for example, is only 50% efficient, while some run-length encoding is almost 100% so. Anyone who wishes to make this kind of tradeoff must take into account whether or not the increase in the price of each node can be offset by savings in the cost of a low-bandwidth transmission line.

Clock recovery and synchronization depend upon encoding schemes. Synchronization is especially important for TDM schemes, since high-speed node clocks need skew for only a few nanoseconds before synchronization is lost. It makes sense to keep these procedures simple and save on processing hardware, even if doing so should require a sacrifice of transmission bandwidth.

Full or half?

Most local networks have half-duplex communications, in the sense that only one packet is on a network at any one time, and every node on it either transmits or receives packets—but not both at once. Full-duplex communications—one channel transmits and the other simultaneously receives—occur on some broadband networks.

Unfortunately, several protocols that are defined for low-speed transmission require full-duplex hardware—independent transmit and receive channels—even though the transmission line is only half-duplex. When the interpacket gap is very small, for example, a node that wants to transmit must do so almost immediately after it receives a packet. Usually, the received packet takes some time going through the node's first-in, first-out buffer, in the funnel. There may not be enough time to clear the FIFO, turn it around to set up the transmit packet, and start transmitting. In such cases as these, an independent channel must be provided for transmission.

An independent channel implies the duplication of the network-interface hardware, which includes buffering, clock synchronization logic, and CRC generation or checking, among other chores. This overhead may be acceptable for low-speed networks, but as data rates increase, the basic cost of the interface hardware goes up, too, and it is not worth doubling the cost to cut the interpacket gap.

The rationale for minimizing the interpacket delay in slow-speed networks comes from the need to utilize the scarce bandwidth of the network as effectively as possible. High-bandwidth networks can afford wasted time on the transmission line; cost and processing overhead are the parameters they must minimize. Consider a typical physical-ring token-passing protocol, like the one defined by the Institute of Electrical and Electronics Engineers' committee 802.5, which requires a node constantly to monitor its transmission rne for a token bit. When this protocol sees one with the priority, it has to modify the bit and attach its own packet within a few bit times.

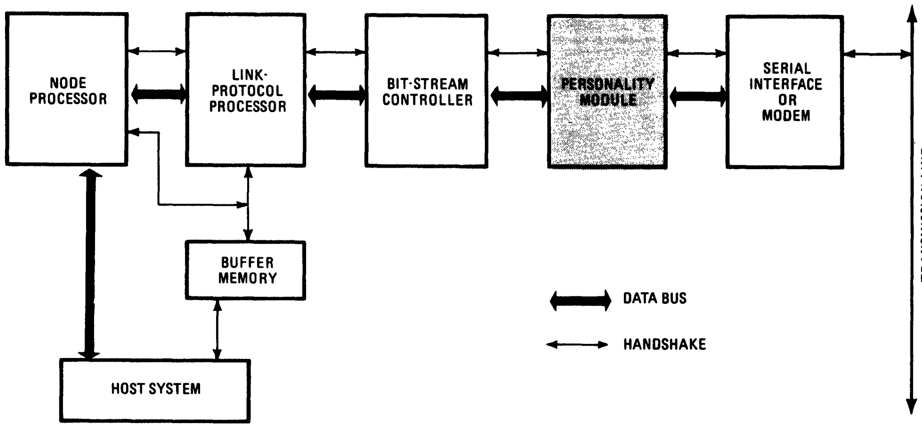
This procedure implies that the node has its own packet prepared and set up, ahead of time, close to the transmission line. For easy implementation at high speed, a simple protocol modification allows certain idle, or fill, characters to be inserted in the transmission line, after the token has been recognized and until the transmit packet is ready for transmission. With this modification, the same hardware can be used for reception and transmission. A protocol being defined by the American National Standards Institute's X3T9.5 committee on the fiber-optic distributed-data interface makes this modification for its 100-Mb/s token-ring network.

Two protocols

Now consider two bus-oriented protocols—Ethernet (Fig. 4), which has been well-discussed, and the ANSI X3T9.5 proposal, which defines a high-speed bus protocol with a specific frame format (Fig. 5) at 50 Mb/s. Carried out with a coaxial-cable transmission line, the proposal would allow as many as 32 nodes to cover a distance of up to 1 kilometer. The access scheme uses time slots for node access, and all nodes have a preassigned time window.

A node whose window comes earlier can transmit before a node whose window comes later. If the earlier node passes up this opportunity, the node with the next window can start transmitting—a virtual token implicitly passed between nodes in the form of a time window. The X3T9.5 proposal permits both priority and equal-opportunity round-robin access.

For the physical layer, the proposal uses a coaxial cable. Instead of encoding the clock and data, it sends them on two independent channels as a broadband signal. Although specified for coaxial cable, the local–distri-



6. The newest. In AMD's 100-megabit/s local-network–controller chips, personality module customizes the interface between the work station and the local network. A link-protocol processor and a bit-stream controller take care of the standard parts of whatever protocol governs.

buted–data-interface scheme is not restricted to this type of transmission line. In fact, this access scheme can be used with fiber-optic cable or even in a star configuration. The number of nodes and the distance covered by the network—both also functions of the physical transmission line—do not affect the access protocol.

Ethernet and the ANSI X3T9.5 proposal are bus protocols that have been optimized for quite different applications. Ethernet is well suited to front-end networks that connect terminals, word processors, printers, and computers. The ANSI X3T9.5 proposal, which works at a much higher speed, is more appropriate for back-end networks, like hard disks and laser printers that are shared by multiple microprocessors.

The table summarizes the differences between the two protocols. The most important differences concern access schemes, transmission lines, header fields, and other such parameters. For sending a response, the ANSI proposal specifies a priority-acknowledgment window to acquire the transmission line. It does not, however, impose a stiff constraint on how soon the response itself must be sent, so it is easy to implement.

It's super

Advanced Micro Devices is not just undertaking a theoretical investigation of local-network protocols for high data rates. It has two different chip sets running at data rates that differ by an order of magnitude.

The first chip set, for Ethernet, is well known. The second, in the early stages of development, is now called Supernet. These chips, though functionally analogous to Ethernet chips, are being designed to provide enough flexibility to implement any protocol or network topology and to operate at data rates up to 100 Mb/s. To achieve that flexibility despite the very great differences among protocols, the functions of the network nodes must be divided between two classes: those common to all networks and protocols and those specific to particular protocols. The Supernet family (Fig. 6) provides a VLSI solution for the common functions and a mechanism that customizes protocol-specific functions as personality modules.

In the configuration represented in Fig. 6, the host can be a microprocessor, a minicomputer, or a mainframe. The node processor, typically a general-purpose microprocessor, takes care of initialization and interrupt handling and could also handle upper-level ISO protocols. The functions of the data-link layer are performed by the link-protocol-processor and bit-stream-controller chips. With the aid of priority-controlled direct-memory-access transfers to and from memory, the link-protocol processor performs functions like address detection and memory-buffer management, and it also generates error status and interrupts to the network controller.

The bit-stream controller, for its part, performs serial-to-parallel conversion of the data that goes to the link-protocol processor and vice versa. It also performs bit insertion and deletion, if necessary, and checks and generates a CRC.

Functions that are protocol-specific have been further divided in two: those that must be implemented in hardware and those that can be implemented in microcode or software in the node processor. More often than not, the access mechanism (TDM, collision-detection, token-passing) must be customized and implemented as a personality module. Users who decide on a simple protocol can implement proprietary protocols with discrete logic or with gate arrays.

The serial-interface adaptor can be an LSI device, or it can be built with discrete components and with specific encoding and decoding of signals. For broadband systems, it would be a modem. □

STANDARDIZING UPPER-LEVEL NETWORK PROTOCOLS

Data communication networks will communicate more efficiently once standards are defined for the upper-level layers of the International Standards Organization seven-layer model.

by David Berry

The need for network interconnection standards has always been apparent. Over the past few years, the emergence of reasonably priced network technology for network connections has increased networking awareness. However, the debate over network technology itself and its access methods—broadband versus baseband, deterministic versus probabilistic, and token access versus carrier sense multiple access—has masked the importance of protocols for data flow control in networking.

As the number of real applications has grown, the International Standards Organization (ISO) has drawn on the experience of proprietary networks, the Advanced Research Projects Agency Network (ARPANET), and the public data network to summarize both known and anticipated networking requirements. ISO has done this by dividing network protocol functions into distinct groupings. This has resulted in a formal specification called the Seven Layer Model. The object of the model is to organize related groups that can be standardized independent of each other. This enables each layer to be defined with minimum reference to any other layer. Thus nodes in a network can be "constructed" with these layers and communication between nodes is ensured via peer-to-peer protocols between each node's layers. (See Panel, "Peer-to-peer is more than just here to here.")

David Berry is section manager of the product planning division, Advanced Micro Devices, 901 Thompson Pl, Sunnyvale, CA 94086, where he is responsible for network protocols. He holds a BS in physics from the University of Edinburgh, Edinburgh, Scotland.

A diagram of a node with the connection to the network medium at the bottom and the user interface at the top (Fig 1) illustrates the model. The functional groups are vertically arrayed.

The primary interface between the classical host functions and the network itself occurs in the network layer (layer 3). This layer allows each host to communicate directly with any other host connected to the global network. Routing and mapping problems are transparent to the network layer service users.

Prior to 1975, layers 1 and 2 were all that were usually specified. Provided that nodes were connected physically on the same network cable, this was adequate. With the development of more sophisticated packet-switching among heterogeneous networks, however, the need for more sophisticated routing protocols became apparent.

The network layer creates a route through the globally interconnected set of networks over which two hosts can converse. To avoid traffic congestion of faulty equipment in nodes of gateways in the path, this may need to be dynamically monitored. There are two basic types of network layers—connection-oriented (CO) and connectionless (CL). The former provides logical or virtual circuits for use by higher layers. The latter deals with each packet independently—ie, a datagram service.

In addition, traffic handling and packet fragmentation/reassembly may be required in order to use available physical circuits. The network layer

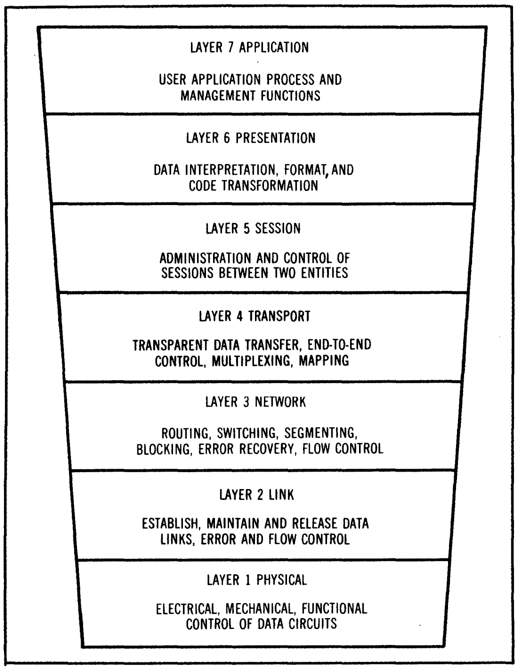

LAYER 7 APPLICATION
USER APPLICATION PROCESS AND MANAGEMENT FUNCTIONS
LAYER 6 PRESENTATION
DATA INTERPRETATION, FORMAT, AND CODE TRANSFORMATION
LAYER 5 SESSION
ADMINISTRATION AND CONTROL OF SESSIONS BETWEEN TWO ENTITIES
LAYER 4 TRANSPORT
TRANSPARENT DATA TRANSFER, END-TO-END CONTROL, MULTIPLEXING, MAPPING
LAYER 3 NETWORK
ROUTING, SWITCHING, SEGMENTING, BLOCKING, ERROR RECOVERY, FLOW CONTROL
LAYER 2 LINK
ESTABLISH, MAINTAIN AND RELEASE DATA LINKS, ERROR AND FLOW CONTROL
LAYER 1 PHYSICAL
ELECTRICAL, MECHANICAL, FUNCTIONAL CONTROL OF DATA CIRCUITS

Fig 1 The seven-layer Open Systems Interconnection (OSI) model developed by the ISO serves to organize communication functions within a network (according to functioning) that can relate to each other. So far, the LAN world has standardized on the two lower levels. The challenge remains to find common goals for standardizing the other five levels.

service is closely related to that of the transport layer above it. (The two were once considered as a single layer.) One of the two layers must provide the virtual circuit over which packets are transmitted between hosts. Both implementations already exist. In public data networks, the X.25 protocol provides for virtual circuits and error detection. In the ARPANET system, the transport layer provides a (CD) service to the user on top of a less complex network layer than the X.25.

Currently, no obvious resolution between the two major types of network service is provided. A comparison of the two is shown in Table 1.

Datagram service is the most primitive of the two. Error and sequence control must be added by the transport layer in order to provide a reliable service to the user. In virtual circuits, much of the service is provided by the network layer itself. This is not always an advantage. When disk sectors are transmitted as packets with an embedded sector address, sequence is unimportant. In this case, delay of data to users because of sequence problems is inefficient and unnecessary.

The network layer itself usually consists of three distinct sublayers, shown in Table 2. Not all three sublayers need be present. Access protocols are not usually required on local area networks (LANs) since LANs tend to provide sophisticated medium access protocols as a part of the data link layer (layer 2). Packet-switched wide area networks (WANs), however, use complex protocols such as the X.25 at this point.

The harmonization sublayer is required only to connect two subnetworks with differing services, such as a CL LAN to a CO WAN. This can be very involved, and practical experience in this area remains limited. The internet sublayer is solely concerned with the problems of mapping and packet-routing between nodes.

Controlling data flow

The transport layer is the highest layer concerned with data movement across the network. It is intended to bridge the gap between the network layer service and the services required by the session layer. Therefore, the transport layer has end-to-end responsibility for reliable data delivery. It monitors and controls all traffic in and out of the node to ensure that no data is lost or garbled. Thus, while the lower layers are concerned with packet framing, transmission, and routing, the transport layer concerns itself with the following areas: virtual connections, the logical circuit between receiving and transmitting nodes; flow

TABLE 1

Virtual Circuit and Datagram Comparison between Network Service

Issue	Virtual Circuit (Connection-oriented)	Datagram (Connectionless)
Initial setup	Required	Not possible
Destination address	Only during connect	In each packet
Error handling	Transparent to host (done in the subnet)	Done by host
End-to-end flow control	Done by the subnet	Not available in the subnet
Packet sequencing	Messages passed in order sent	Messages passed in order received

TABLE 2

Network Sublayer Functions

Sublayer	Name	Function
3a	Access	Provides the interface to the data link layer
3b	Harmonization	Provides a mapping of the services available to give a common set to sublayer 3c
3c	Internet	Provides the routing and address mapping required to cross the network

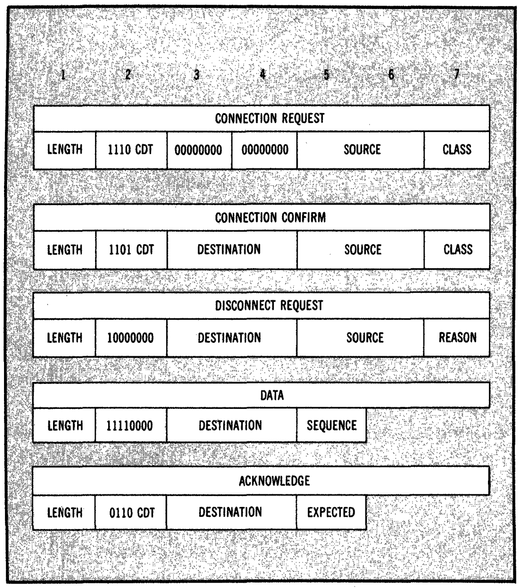


Fig 2 The transport level is the highest level that concerns itself with the movement of data across the network and where the real problems of the OSI must be solved. The OSI transport protocol standard headers shown here are compatible with most major standards organizations.

control, monitoring the amount of data not yet acknowledged as received; error detection, the ability to monitor the packet sequence for missing, duplicated, or missequenced packets; and error recovery, the ability to rectify errors detected by this or lower layers without host intervention.

To a large extent, the degree of complexity required in the transport layer depends on the quality of the network across which the data is being moved. Reliability may be a function of the network itself, but it may also depend on the sophistication of the protocols being run at the network layer. To reflect this, transport service is generally made available in one of five classes: Class 0—simple, for use with Comite Consultatif Internationale (CCITT) networks; Class 1—basic, not yet fully defined; Class 2—flow control, limits number of outstanding acknowledgements for packets sent; Class 3—error detection, detects mishaps in the flow; and Class 4—error recovery, corrects mishaps automatically.

TABLE 3
Transport Layers

Standards Body	Transport Protocol Standard
ISO	DP 8072
ISO	DIS 8073
ECMA	72
DoD	TCP
CCITT	So70
NBS	TP

The transport service provides a pair of queues between two users that carry both data and control information in a duplex manner. It provides this as a transparent service, which then removes any need for users to be concerned with the physical data transmission.

Examples of transport layer protocol implementation are the transmission control protocol used in ARPANET, the network services in DECNET, and transport control systems in systems network architecture (SNA). None of these is normally used with a CO network layer, as in an X.25 network. This highlights the main division in the current debate on the transport layer. An example of the headers used in the ISO transport protocol standard are shown in Fig 2.

With transport layer protocols, two different approaches have been taken. On the one hand, packet-switching proponents, headed by CCITT, have approached networks from the point of view of existing telephone networks. Much of the work to be done in the transport layer is already incorporated in the virtual circuit protocols of the X.25 network layer. Some vestigial functions are required, such as reset recovery, since the X.25 protocol allows asynchronous resets to the connections.

On the other hand, those involved with computer networks require more transport layers. The main driving forces behind this approach were the National Bureau of Standards (NBS) and the ARPANET community in the U.S., and the European Computer Manufacturers Association (ECMA) in Europe. The majority of computer and system manufacturers are supportive of their efforts for a more complex protocol since this is the layer at which the real problems of open system interconnection must be handled. The major standards at the transport layer are listed in Table 3.

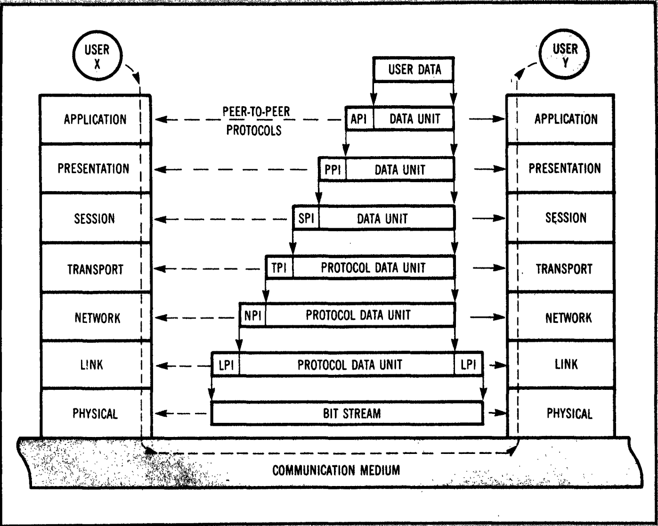
Although Table 3 seems to imply a variety of standards, the ISO, ECMA, and NBS standards are in fact virtually compatible. At the time of writing, a single transport standard will shortly be ratified by ISO. This is based on the NBS/ECMA standards (which are compatible in all but some details) and includes a class that covers the CCITT requirements for a truncated transport protocol.

In the attempt to define protocols, the transport layer appears to be the highest layer at which it will be possible to find common ground among all applications for a single standard. This explains the comparative rapidity with which a common standard for this layer has been achieved.

Making the connection

The session layer functions are still the subject of major discussion in networking. Several existing networks (such as the ARPANET) dispense with the

Peer-to-peer is more than just here-to-here



The seven-layer OSI model serves as a road map for network users. Network communication begins when the user in one node addresses the top or seventh layer of that node. This layer then communicates with the corresponding layer in the other user's node with some form of peer-to-peer protocol. This is accomplished by having the seventh layer use the services of the next lower layer, much as the user makes use of the seventh layer. This lower layer, in turn, corresponds with its opposite peer layer by means of its own peer-to-peer protocol, which is transported using the service of the next lower layer.

The importance of this layered arrangement is that functions associated with a network connection are grouped in an ascending order with respect to the host, and in a descending order with respect to the medium. The decoupling of layers allows a wide degree of flexibility to the network system designer in choosing the functions needed layer by layer. Each functional group is concerned only with those tasks associated with the layer in question. A node can therefore be assembled in a highly modular fashion.

Functions that are associated with lower layer protocols include packet framing (grouping of characters and messages), error control (detection of erroneous data or formats), data encoding (bit coding of the characters being sent), and frame addressing (routing of frames to destination nodes). These functions are typically concerned with providing access to the transmission media.

Besides the required protocols to communicate between the same layer in different nodes, protocols are also needed for communication between different layers in the same node. One layer interacts with another via a protocol in three ways: peer-to-peer, where a layer communicates with the same layer of another node; service provider, where a lower layer provides a service to the next higher layer of the same node; and service requester, where a higher layer uses the services provided by the next lower layer of the same node.

To date, most of the network protocols that have been considered for standardization are peer-to-peer protocols. Information is passed from peer-to-peer by attaching a header to the beginning of the data packet that is to be transmitted. Each layer encodes its header according to its specification (see Figure.)

Each layer provides a service to the layer above and expects service from the layer below. It receives data units from the layer above, along with interface control units that indicate the service to be performed for the layer above. The data units will already contain appended headers with protocols from the higher layers. This is completely transparent to the lower layer—ie, the lower layer has no understanding of the protocol used by the preceding layer and has no way to distinguish upper-layer protocol headers from the data proper. The lower layer then operates on the data packet as required. It might, for example, encrypt the data or split the data into packet lengths acceptable to the network to be traversed. Finally, it will pass control to the next lower layer by means of another interface control unit. The header now appended to the packet is transparent to the next lower layer.

Upon receipt of the packet at the destination node, a similar process is performed in reverse order. Each layer examines the packet area where it expects to find its header. This is interpreted according to the protocol specification for that layer, and any remaining part of the packet is passed up to the next layer for further examination as required.

Not every packet originates or terminates in the application layer. Also, several physical packets are usually required to transfer one logical packet. The actual ratio depends on a number of factors, such as the protocols in use, the error rate of the network, and the number of fragments into which a packet must be broken.

The logical mechanism by which protocols are transmitted across the network is a set featuring commands known as primitives. These are encoded in the appropriate packet header. Each primitive is generated by a layer within a node and transmitted to its peer in the other node by means of the services provided by lower layers. A given layer invokes these services by passing a primitive, such as a REQUEST, to the lower layer. This appears in the destination node as a REQUEST being passed up to the peer layer by the lower layer. Responses are passed back in a similar manner in the opposite direction. Thus, orderly transactions are ensured.

session layer altogether. The basic session layer functions manage data transfer (using transport service to move raw data), and add three categories of user-oriented services: connection management, data transfer, and transfer management.

Connection management provides such features as remote user identification that allows the local resident operating system to determine user privileges at the remote node. Data transfer service provides simultaneous bidirectional data transfer between nodes. Transfer management provides some form of subunit synchronization within the entire data block to be transferred, permitting recovery from network errors without retransmission of the entire file. Dialogue between the session entities of two nodes proceeds as an interchange of data units via the transport service (Fig 3).

Discussion of the session layer centers around four main philosophies: the ECMA approach, as embodied in the ECMA-75 standard; the CCITT approach, as embodied in the S.62 standard; the American National Standards Institute (ANSI) approach, a simplification of both; and the ISO working draft, a combination of the ECMA and CCITT approaches.

The ECMA approach is based on the software architecture of mainframe session functions. It attempts to provide the user with a more generalized control of the transport services of the layer below. The functions provided include reliable, organized, and synchronized data transfer; four distinct service subsets; and an optional data quarantine function.

In addition, the commands that are used by the protocol can be classified into four groups: session (connect/release/abort); data (data/expedited); synchronization (sync/resync/endDU/tokens); and quarantine (deliver/cancel).

The CCITT approach is based on the requirements of the teletext service and makes a number of less general assumptions about the application than the ECMA approach. It is designed to operate on a public data network and assumes that the major use will be in a teletext environment. It has only one distinct class of operation. Within this class are a number of possible commands, divided into two groups—session commands and document commands. These groups correspond roughly to the session and synchronization commands of ECMA-75.

The ISO approach, as characterized by the last meeting of the TC97/SC16/WG6 committee in Mar 1983, has been to combine all of the facilities of both the ECMA and CCITT approaches. This has resulted in a somewhat cumbersome specification that includes all of the command types from ECMA-75, plus an activity management (ie, document control) command group, plus exception reporting, capability data exchange, and typed data.

This complexity has not yet been well received in the U.S. and ANSI has yet to come to a definitive

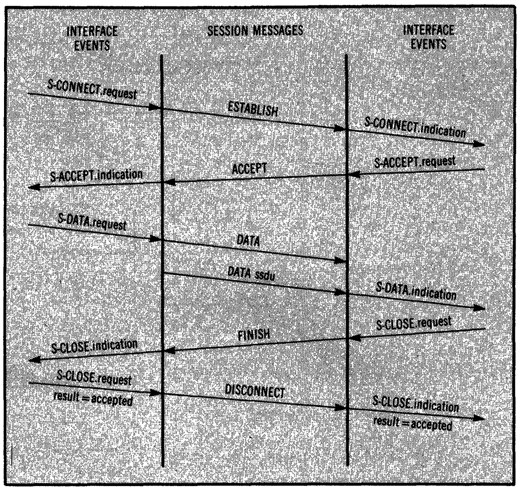


Fig 3 The session level manages the data that the transport level transmits in an unmessaged state. The dialogue between two session levels follows four different approaches advocated by the following organizations: ANSI, ISO, CCITT, and ECMA.

position. The group has neither issued a draft standard of its own nor modified one of the above. There is unlikely to be agreement on the basis of any of the above due to the feeling among major U.S. manufacturers that the session layer does not require this degree of complexity and corresponding overhead. The likelihood of a unified session standard being available in the next year is not good. It may even prove necessary to agree on multiple standards that are appropriate for differing applications.

Speaking the language

While layers 5 and below deal with the transmission of data units that are transparent to them, the presentation layer is more concerned with network user interface. As the name suggests, the presentation layer presents the user with data transferred on the network in an orderly and unambiguous manner.

This layer is therefore application-specific, and any comprehensive standard that will define all its functions in one protocol is unlikely. Currently, a number of nonoverlapping protocols for the presentation layer are being defined. These include virtual terminal functions; video/teletext; packet assembly/disassembly (PAD) functions; and text compression and character coding.

The most comprehensive standard for the presentation layer so far is the ECMA-86 standard on generic presentation service (GPS). The standard describes a model of the layer and the services it provides. It defines the GPS as a set of common service facilities that are dependent on the underlying session service to establish and maintain contact between users of the presentation services. Once contact has been established, all communication between the users takes place according to the GPS protocol, known as the p-service. Access to the p-service by the user is performed by an implementation-dependent interface.

Within the GPS, many service subsets are possible. These correspond to applications, such as file transfer or data base management. A specific presentation service and a protocol used to provide it should be defined for each application.

Virtual terminal protocols (VTPs) are a subset used at the presentation level. An example is the ECMA-88 basic class virtual terminal, which is a subset of the ECMA-87 standard generic virtual terminal description of a virtual terminal service. Virtual terminals are a convenient construct in networking.

In practice, both the terminal and the application have access to the data being transferred as a two-dimensional array. The array consists of a number of cells that can be accessed (read from or written to) by either. The main difference is that the application has direct access to the data structure, while the terminal user views it through the display device and modifies it using the keyboard.

The application and the terminal communicate by means of character strings. A string is entered via the keyboard, which is passively viewed by the application until it can be interpreted as a command or response. The application then takes over and performs the appropriate action. The result of this is then communicated to the operator by means of a message entered in the data structure.

Since the terminal and the application both have their own views of the data structure, it is the task of the virtual terminal protocol to synchronize their communication. It negotiates the compatibility of the data structure, and establishes a set of functions to be implemented in a standard fashion. However, it does not address the exact nature of the terminal itself.

In addition, the upper interfaces of the VTPs to the terminals concerned are nonstandard. However, the interface to the session layer and the protocol used to communicate with another node follow the VTP standard.

Teletext and videotex are examples of presentation layer functions. Both have their origins in the idea of a low cost information service available to both the home and the business market. Information in the form of both text and graphics is disseminated by the network to some kind of display, usually a modified TV set.

The distinction between teletext and videotex is in the degree of user interaction. Teletext permits almost no control over the selection of the data transmitted. A number of information screens are repeatedly transmitted, and the user may select one to display. Videotex users equipped with a small keyboard can select from the data base available on the network. In real applications, these distinctions become blurred, and teletext and videotex are generally grouped together. Moreover, they provide a spectrum of services ranging from one-way transmission from dumb terminals up to enhanced services available on sophisticated two-way transmissions from external data networks and intelligent terminals.

Examples of teletext in use today are the Ceefax and Oracle systems provided by the two television networks in Britain, and the Antiope in France. The current market in Britain is the largest in the world for teletext and provides 100-page magazines via the blanking intervals (the period of time when the electron beam is being repositioned at the top of the screen for another pass) in ordinary TV broadcasts.

Both teletext and videotex point the way to electronic information delivery to the home and office as a matter of routine. Limited bandwidth broadcast systems are excellent media for news, sports, and entertainment guides. More sophisticated interactive systems can also offer teleshopping and electronic banking.

Both systems require that a universal standard for transmission of the information is available. This is less of a problem with text displays than it has been with graphics. However, within the last year, there has been a considerable increase in support for the North American Presentation Layer Protocol Standard (NAPLPS). This is a method for encoding visual information in a standard and compact manner. Graphic and textual information can be represented in a variety of modes, colors, and styles. In addition, facilities are provided by the protocol for the user to interact with the two-dimensional display in a very flexible manner.

Limited bandwidth broadcast systems are excellent media for news, sports, and entertainment guides.

NAPLPS is one of the three contenders for a presentation layer standard for a graphics protocol. Whether the European Conference Éuropean des Postes et Télégraphique (CEPT) or the Japanese Captain will be incorporated in, or simply supplanted by, NAPLPS remains to be seen.

The PAD functions of the presentation level protocols historically preceded the ISO model. For that reason, PAD is something of a misfit in that it belongs conceptually in the presentation layer, but operates as if it were a part of the network layer.

The PAD was a parametric approach to a virtual terminal taken by the CCITT in their efforts at standardization. It contrasts with the virtual machine approach described above by using a predefined set of parameters to define all terminal characteristics.

A set of these parameters is negotiated for each connection made. This is easy to implement, usually as a table-driven scheme. Additionally, the standards in general use for these PAD functions are all from the CCITT "X" series. They include X.3, X.28, and X.29.

The text compression and character coding parts of the presentation level protocol are terms used to describe the encoding of data to be transferred on the network and its subsequent decoding at the receiving node. This can usually be related to encryption.

Text compression attempts to reduce the amount of traffic on the network by encoding repetitive strings of characters—eg, by putting blanks in a more compact form. The strings are then expanded to their original format at the destination.

Character coding is used when communicating between hosts that use incompatible coding for their data, such as extended binary-coded decimal interchange (EBCDIC), and ASCII 7 bits. There are few true protocols involved in either character coding or text compression, and little has been done toward universal standardization.

Talking to the user

Of all the layers involved in networking, the application layer provides the most diversity and yet the least opportunity for standardization. It is the one most intimately concerned with user processes. The term "user" is generally taken to mean any user process or user machine, not necessarily a person using the system directly. The diversity is particularly true in the present environment where a multiplicity of systems exists.

The boundary with the presentation layer separates network designers from true network users. This layer has received less detailed attention than any of the layers, in part because of the lack of motivation towards standardization. The activities associated with the application layer are usually considered to include file transfer control; data base management; remote job entry (RJE) and data handling; electronic mail; and network operating systems.

File transfer protocols (FTPs) handle files that are accessed by remote users in a standard fashion. This differs from local file management provided by the operating system. The operating system creates files with unique names within the local system and manages them according to local conventions. For global networks, a network-wide file management system needs to be defined. In the local environment, the most common operation is file access. On the network, it is file transfer.

The criterion for file transfer is to establish network-wide conventions for unique file naming. The FTP identifies both the source and the destination, and locates the file(s) involved. Details such as privilege of access, code encryption, or compression and merging must be standardized.

After the initial control phase, the data transfer takes place with associated control between the two servers involved. In the transfer of long files, the session layer services can be vital to efficiency. Should an error occur during file transfer, the session protocol should be capable of establishing the most recent checkpoint up to which the data was correctly transferred and of restarting the transfer from this point, rather than from the beginning of the file.

The application layer provides the most diversity and yet the least opportunity for standardization.

Currently, the closest thing to a standard in this area is an ISO FTP. It includes a logical model to help define the concept of virtual filestore that enables access and file management in the global network as well as the local environment. As a result, nonhomogeneous processes will be able to function interactively.

Database protocols are a generic grouping that includes such specific examples as RJE and interprocess communication protocols. The need for data bases was advocated long before the advent of networking and the availability of lower cost distributed systems. Many applications have multiple geographic locations where data must be available and operated on. Relational data bases have evolved to meet the requirements of query processing, concurrency control, and the relational distribution problem. None of this has yet been standardized for the networking environment.

Communication paths between data bases can either exist within a single-user system, or can be used to connect several user systems over some form of network. In a distributed system, the distinction between a remote and a local connection becomes trivial. Any protocol standard for distributed data bases must control both the access and any associated communication in an orderly manner. Processes can be located either in a single system or across several systems. In the latter case, the liaison must be established in order to exchange information, whereupon it makes no real difference whether the process accessed is local or remote.

The RJE protocols are used to enter jobs or batches of jobs to a remote computer. These will access the network through some form of VTP at the presentation layer. It requires a degree of high level activity, with program files being transferred by an FTP. Little has yet been done to achieve standardization in this area.

The details of how FTPs, distributed data bases, and RJEs will interact in the networks of the future

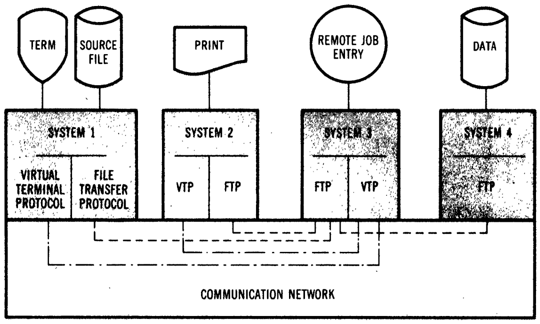


Fig 4 The application level is concerned with the end-user applications and is the least likely to become standardized. Five generic application activities are handled at this top level: file transfer control, database management, RJE, electronic mail, and network operating systems. For the most part, the transactions are handled through file transfer protocols (FTPs).

is a matter of speculation. A general scheme is illustrated in Fig 4.

Electronic mail is the means whereby messages may be exchanged by the same network users. This has received considerable attention recently. It is also known as computer-based message systems and is conceptually divided into three sublayers within the application layer: user agent sublayer, which provides message preparation tools; message transfer sublayer, which provides transfer mechanism; and reliable transfer sublayer, which controls error recovery. While a number of vendors

The remote job entry protocols are used to enter jobs to a remote computer.

such as MCI Communications Corp (Washington, DC) provide electronic mail service, little progress has been made towards a standard.

Business data exchange is an area in which ANSI is attempting to reach a standard protocol agreement. The series of X.12 standards attempts to define protocols for the following common business issues: purchase order transactions (X12.1).; invoice transactions (X12.2); and data elements (X12.3). Although these three standards have been officially approved, they have not yet been applied widely.

Network operating system is a generic term for the concept of distributing the functions of an operating system among the nodes of a network. It is a logical extension of hardware distribution implied by networks to the software itself.

Each host continues to run its prenetworking operating system, but the network operating system is implemented as a collection of user programs distributed among the nodes that manage data and communication in an orderly and uniform manner. Little has been done in this area, but the problems are closely related to those of a distributed data base. Thus, the functions of a node can be efficiently organized by adhering to the specifications of the seven layers of the OSI mode.

STEP MOTORS

Design real-time Ethernet systems by giving priority to nodes

You can eliminate the transmission uncertainties of Ethernet for a few critical nodes if they are given special access rights to the network using simple design rules, two extra gates and a timer.

Vernon Coleman, *Advanced Micro Devices*

If you are designing a real-time system, you can use some simple priority schemes to enjoy the benefits of low-cost Ethernet hardware and software by addressing Ethernet's indeterminate response times. These response times are indeterminate because the network gives all electronic traffic between nodes the same priority. This scheme, dubbed fairness, limits the effective data-transmission rate to about 9 MHz.

While the fairness approach works for most networks, there are important exceptions—real-time or time-critical systems—in which individual nodes on the network must be able to send data rapidly without an indeterminate waiting time. The priority schemes described in this article can help you realize the transmission speeds those special nodes require.

Be "unfair" to improve real-time operation

Nodes on the network that need high bandwidth might be impeded under Ethernet's access scheme. (If you are unfamiliar with Ethernet's network-access scheme, which is based on CSMA/CD and a binary exponential backoff algorithm, see **box,** "An equal-opportunity network.") Because these high-bandwidth nodes must respond quickly, they need an upper bound on network access time; they must be able to transmit within a certain time limit.

One example of a time-critical node is a file server, which is typically a high-performance storage system accessed frequently by other nodes on the network (**Fig 1**). Nodes generally send short requests to the file server and in return receive large files of data contained in many packets. Because, in many cases, the accessing nodes can do nothing until their queries are answered, the server, whose priority is equal to that of any other node, becomes a bottleneck; it can't get onto the network because other nodes are asking it for files. It is therefore critical that a server have higher bandwidth than other nodes under all network load conditions.

Make one node the master

You can implement either of two types of priority to increase a node's bandwidth: exclusive priority or group priority. Exclusive priority ensures that one and only one node—the master node—has the authority to transmit when transmission contention exists. In an Ethernet system, you implement exclusive priority by inhibiting all nodes from transmitting—except the master node—if they sense a collision when they try to transmit.

Calculate the maximum transmission time by first identifying the worst-case total time to transmit a packet. Six factors define this time:

- T_{ECT}—the time for the current transmission to end (0 μsec for back-to-back transmissions, 1210 μsec otherwise)
- T_{WBT}—the waiting time before transmitting
- T_{BDC}—the time before detecting collision
- T_{BT}—the backoff time
- T_{WBR}—the waiting time before retransmitting

- T_{TP}—the time to transmit a packet (51 to 1210 μsec depending on packet size).

T_{WBT} and T_{WBR} each equal the interpacket-gap transmission time, defined by the Ethernet specification as 9.6 to 10.4 μsec. T_{BDC} and T_{BT}, on the other hand, each equal the minimum slot-time interval, given in the spec as 51.2 μsec.

To calculate the exclusive-priority bandwidth (B_{EP}), you add the above components of delay and divide them into the length (L_P) of the packet to be transmitted:

$$B_{EP} = L_P/(T_{ECT} + T_{WBT} + T_{BDC} + T_{BT} + T_{WBR} + T_{TP}). \quad (1)$$

Depending on packet length, and whether there is back-to-back or intervening transmission, this equation yields B_{EP} values from 370 kHz min to 9 MHz max.

Give a group of nodes priority

Exclusivity naturally carries the limitation that only one node on the network can transmit with priority. What if you want to have more than one file server with such transmission privileges? In this case, you must implement a group priority.

Unlike exclusive priority, the group scheme does not give an absolute lower bound for network-bandwidth allocation. Instead, group priority greatly enhances the probability that a node belonging to that group will be able to transmit within a given time window. To understand and implement successfully a group-priority scheme for your network, you have to know the probability for successive collisions. Once you know this, you can determine the transmission delay for various numbers of nodes in a group.

Assume there are S possible slot times available for the start of a transmission and N nodes which will begin transmitting in a random slot time. The probability of no collision for:

- The first node=(S empty slots)/(S total slots)=S/S.
- The second node=(S−1 empty slots)/(S total slots)=(S−1)/S.
- The third node=(S−2 empty slots)/(S total slots)=(S−2)/S.
- The Nth node=S−((N−1) empty slots)/(S total slots)=(S−(N−1))/S.

Consequently, the probability for no collisions (P_{NC}) for N nodes in S slots is:

$$P_{NC} = (S/S)((S-1)/S)((S-2)/S)\ldots((S-(N-1))/S), \quad (2)$$

or, simplifying,

$$P_{NC} = S!/(S-N)!/S^N = S!/(S-N)!S^N. \quad (3)$$

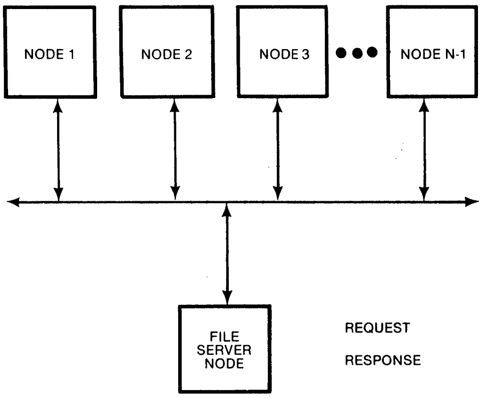


Fig 1—The file server bears an unequal share *of the load in the Ethernet-linked system shown above. When queried—usually with a small, single packet—its typical response is to send multiple packets, which compose the entire file or files. If the file server has no priority, users might be idle for excessive time while waiting for a file.*

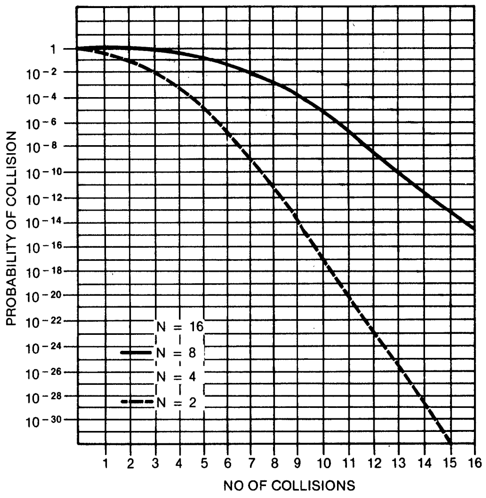


Fig 2—The probability of collision *for a network with group priority depends upon the number of nodes in a group and how many collisions the user can tolerate in a time-critical system. Use this chart to determine the best tradeoff for your design.*

An equal-opportunity network

The Ethernet specification, meeting the applicable criteria of IEEE Standard 802.3, mandates that nodes be arranged in what is commonly known as a bus or linear architecture, and that they also operate without priority, ie, with "fairness" for nodes. This fairness doctrine uses two techniques in conjunction: carrier-sense multiple-access with collision detection (CSMA) and binary exponential backoff.

With CSMA/CD, a node can access the network if it does not detect that another node is transmitting. It senses the energy output, referred to as the "carrier," of other network nodes. (The energy output is called the carrier because CSMA/CD was first used in a broadband system called ALOHA. In ALOHA, stations sensed the modulated carrier transmitted by other stations in the network to determine if they should transmit.) If a node wants to transmit on the network but senses a carrier, the node will defer its transmission until the carrier goes away.

The node is free to attempt another transmission 9.6 μsec after a previous transmission ceases (a delay called the interpacket-gap time). However, two or more nodes could begin to transmit simultaneously after sensing that there is no carrier on the network. In this instance, the transmissions from the nodes will collide, and all the information will be scrambled. The nodes sense this collision as a rise in energy on the network above that for a normal transmission.

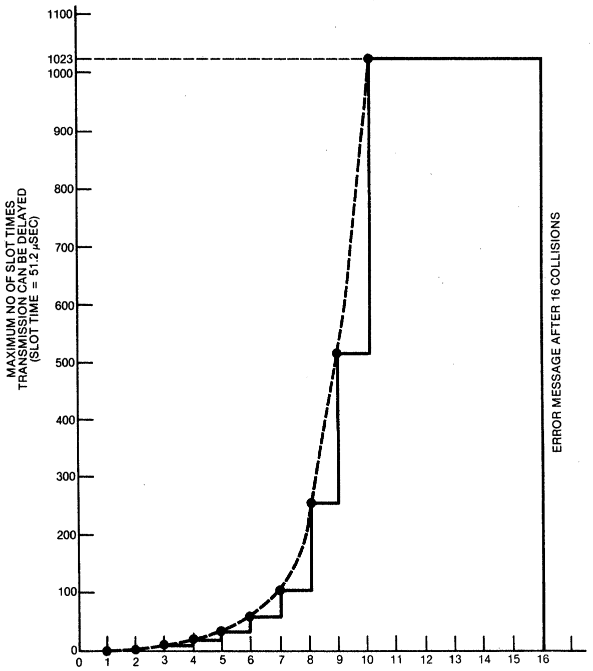
When they sense a collision, the transmitting nodes start executing a load-shedding algorithm called the binary exponential backoff algorithm. It ensures that only one transmission occurs at any given time. The algorithm, implemented in each node, provides this insurance by having a would-be transmitter node "back off" for a random period of time before attempting to transmit again.

There are progressive limits to this random period. The limits consist of multiples of a slot time, which the Ethernet spec defines as 51.2 μsec. This value allows the nodes that are the greatest distance apart on the network to detect collisions.

In accordance with the spec, the backoff algorithm permits a maximum of 16 collisions for any given transmission attempt. After 16 collisions, the node signals an error. The slot-time multiple increases by a power of 2 (hence *binary* backoff) after each successive collision (see **figure**).

In other words, after one collision the maximum waiting time is one slot time of 51.2 μsec (the multiple is binary 1). After two collisions the maximum waiting time is three slot times (binary 11). After three collisions the maximum time is seven slot times (binary 111), etc. The maximum waiting time reaches 1023 (binary 1111111111) slot times after 10 collisions and remains at that value until the 16th collision. Because these are maximum waiting values, the actual waiting times for each of the nodes involved in the collision will vary randomly between zero and the maximum time.

The randomness associated with the binary backoff assures that each of the nodes involved in a collision will have an equal opportunity to access the network eventually. Equal-opportunity access also means that it is only by pure chance that any node will perform a successful transmission within a specified time limit. It is this uncertainty of transmission that could cause problems in certain applications.



The result of a binary exponential backoff algorithm *is shown graphically here. As Ethernet specifies, each time a collision occurs, all nodes trying to transmit will back off for a random amount of time within a fixed time limit that grows with each collision.*

Ethernet in VLSI

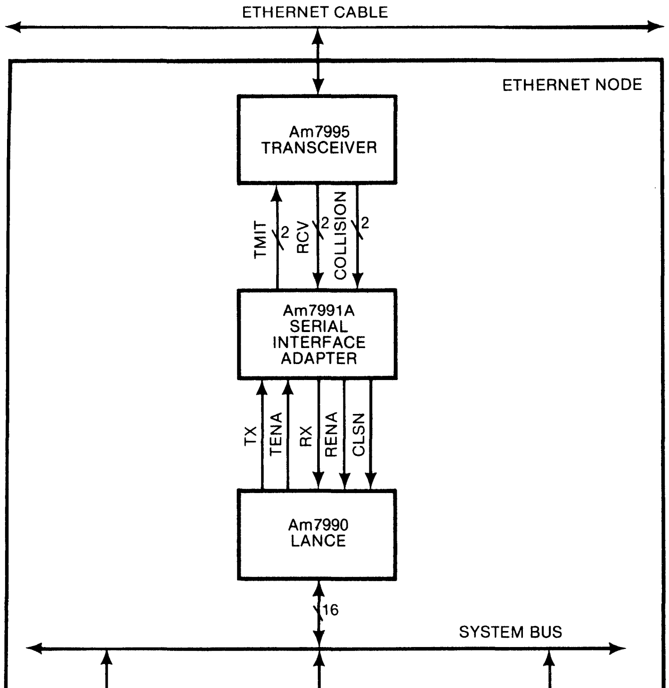
The Ethernet implementation described in this article involves two chips of a 3-chip set. The two involved are the Am7990 local-area-network controller for Ethernet (LANCE) and the Am7991A serial-interface adapter (SIA). The third chip of the set is the Am7995 Ethernet transceiver. Typically, the three chips connect a node to a network in the manner shown in the **figure.**

The Am7990 LANCE is the primary link between the node's central processor and the Ethernet system. It enforces the CSMA/CD access protocol, manages external memory for transmission and reception, and reports errors. The Am7991 SIA encodes and decodes data going to and coming from the main coaxial cable using Manchester encoding and decoding. It also converts the TTL I/O of the LANCE to the differential I/O of the transceiver. For incoming data, the SIA recovers the clock with an onboard phase-locked loop; for outgoing data, either an external crystal oscillator or timing inputs provide the clock.

The Am7995 transceiver complies with Ethernet specifications, but it also offers the option of operating in networks with longer electrical lengths. It contains jabber-control circuitry as well.

To transmit a packet, the LANCE initiates a DMA cycle and pulls data from a transmit buffer set up in system memory. It frames the data by prefacing it with a preamble and sync pattern and appending a 32-bit cyclic redundancy check (CRC), which the LANCE calculates. It transmits the packet with CRC serially to the SIA, which in turn creates the Manchester-encoded differential signals to drive the transceiver. The transceiver then converts this signal to a single-ended drive on the coaxial cable.

When it receives a packet, the transceiver creates differential output signals to drive the SIA's inputs. These inputs to the SIA are decoded by its Manchester decoder, and its phase-locked loop synchronizes to the incoming packet's preamble, allowing the decoder to recover clock and data from the encoded signals. In addition, the SIA creates a receive-enable signal while it is receiving, indicating to the LANCE that received data and clock signals are available. When the packet reaches the LANCE, the preamble is stripped and the CRC is checked for correctness. The packet and its associated status are automatically stored in receive buffers in main memory.



This VLSI chip set forms a complete link *between a system (node) and the Ethernet. The VLSI set reduces the design time needed for implementing the network protocol.*

The probability of collision (P_C) for N nodes with S slots is

$$\begin{aligned} P_C &= 1 - P_{NC} \\ &= 1 - (S!/(S - N)!S^N). \qquad (4) \end{aligned}$$

You can now calculate the probability of successive collisions (P_{SC}) with N nodes and a number of slots, which, according to the Ethernet spec, increase by the power of 2 (to a maximum of 1023) after each collision, for as many as 16 collisions:

$$P_{SC} = \overbrace{(1 - (2!/(2^1 - N)!2^1))}^{\text{1st collision}} \times \overbrace{(1 - (2^2!/(2^2 - N)!2^2))}^{\text{2nd collision}}$$
$$\ldots x \overbrace{(1 - (2^{10}!/(2^{10} - N)!2^{10}))}^{\text{10th through 16th collision}}. \qquad (5)$$

Note that, if N>S, then P_C=1 using **Eq** 4. This validates the intuitive observation that, if there are more transmitting nodes than slots, there's bound to be a collision.

Fig 2 shows in graphic form the results of evaluating **Eq** 5 for values of N from 2 to 16. Note the sharp reduction in the probability of successive collisions as N decreases. It is this reduction that you can capitalize on in the group-priority scheme.

Use group priority to optimize bandwidth

An example helps show how establishing group priority optimizes a priority node's potential bandwidth. Using **Fig** 2 as a guide, you first choose a small priority group. Assume for this example that the group has two member nodes that impose equal loads on the network. Next you choose a minimum probability of collision resolution for the priority group. For this example, choose 1×10^{-3}. The curves in **Fig** 2 show that 1×10^{-3} intersects the curve for two nodes close to the point of four collisions. This represents one chance in one thousand that a collision among a priority group of two nodes will not resolve in four attempts or fewer.

Now you can find the bandwidth for a priority transmission that has two numbers, each transmitting one packet. Divide the lengths of the two transmitted packets by the maximum time for transmitting them. The maximum time for transmitting the packets comes from **Eq** 1 and contains additional terms to account for multiple retries (M) and the delay time to transmit the second packet of the set. M is simply the number of collisions (four) dictated by your choice of P_C.

Because the maximum number of slot periods that must pass before attempting to transmit increases by a power of 2 after each collision (per the Ethernet specification), the priority group's bandwidth (B_{GP}), including the eventuality of priority-transmission collision, is:

$$B_{GP} = 2L_P/ (\overbrace{T_{ECT} + T_{WBT} + 2^{M-1} (T_{BDC} + T_{BT} + T_{WBR}) + T_{TP}}^{\text{delay for the first packet}} + \overbrace{T_{WBT} + T_{TP}}^{\text{for second packet}}). \qquad (6)$$

For this example M equals 4. Consequently, for L_P set to a maximum packet of 12144 bits, B_{GP} equals 454 kHz. For a minimum packet size of 512 bits, B_{GP} equals 340 kHz. As defined by **Eq 6,** if there is only one packet instead of two transmitted with priority, the priority bandwidth is halved, because $2L_P$ becomes L_P.

You can keep group-priority bandwidth high by restricting the group's size to two to three members. Increasing this number can have adverse effects on nonpriority nodes because of network-access delay, as the denominator of **Eq 6** shows.

Implement priority in hardware

To implement group priority in hardware, you must add extra circuitry to each nonpriority node on the network. The same circuitry can also be added to the priority nodes, so that you have flexibility when configuring a system. To show how this can be easily done,

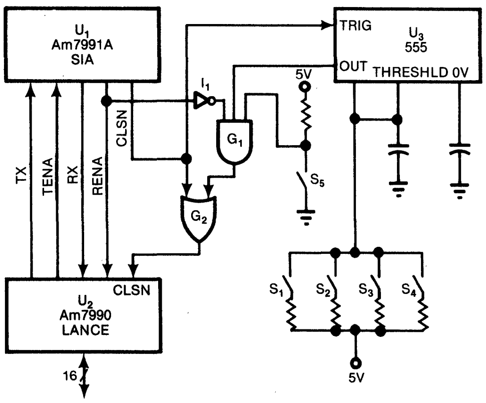


Fig 3—To build a priority system, *add this circuit to each node in the system. The resistor network controlled by switches S_1 through S_4 programs the duration of the timer's output. A nonpriority node's timer is set so that the priority node or nodes have sufficient time to get a message onto the net before the nonpriority nodes try to transmit.*

the following example uses two chips of a 3-chip set (see **box**, "Ethernet in VLSI"). **Fig 3** shows a circuit which can implement either exclusive or group priority.

The CLSN input to the local-area-network controller for Ethernet (LANCE) prevents it from transmitting packets. For any node, regardless of priority, you want this input active whenever there is a collision. For a nonpriority node, you also want CLSN to be High when there is no activity on the network, but a priority node might be waiting to transmit.

In circuit shown in **Fig 3**, if CLSN from the serial-interface adapter (SIA) goes High, the output of the 555 timer will go High (**Fig 4**). You must program the 555's output to remain High for the time calculated so that the priority node can transmit. How you arrive at this time depends on whether you choose an exclusive- or a group-priority scheme.

If you use an exclusive-priority design, the timer will be set to two slot times plus an interpacket-gap time ($T_{BDC}+T_{BT}+T_{WBR}$ of **Eq 1**). This equals 112.4 μsec. For group priority, you set the timer for the worst-case sum of the backoff and waiting times needed to resolve collisions between two priority transmissions. For our example the timer setting comes from **Eq 6** and equals $(2^{M-1}(T_{BDC}+T_{BT}+T_{WBR})+T_{TP})+T_{WBT}+T_{TP}$, or 4.1 msec. Setting switches S_1 through S_4 programs the 555 for different output-pulse durations.

You could tie the timer's output to the LANCE's CLSN input, except that when there is a transmission, you want all nodes to receive whether they have priority or not. When the LANCE's CLSN line is High, it can neither receive nor transmit.

To control the CLSN input to the LANCE so that it can receive data, you must include an AND gate (G_1), an OR gate (G_2) and an inverter (I_1). Together, the gates permit the timer's output to raise the LANCE's CLSN line when receive enable (RENA) is Low. By inverting the output of the SIA's RENA line and feeding it to the AND gate, the RENA signal can turn off the CLSN input to the LANCE, allowing it to receive data. **Fig 4** shows the waveforms that result.

If you wish, you can install this circuit in all network nodes and change the priority scheme as you see fit. To turn a nonpriority node into a priority node, you close switch S_5. Closing the switch causes one input to the AND gate always to be Low; therefore it's output can never go High and send a CLSN signal to the LANCE. With S_5 closed, only the CLSN signal from the SIA will disable the LANCE. The switch, cutting out as it does all effects of the priority circuit, also permits you to move nodes from a prioritized network to a nonprioritized one. **EDN**

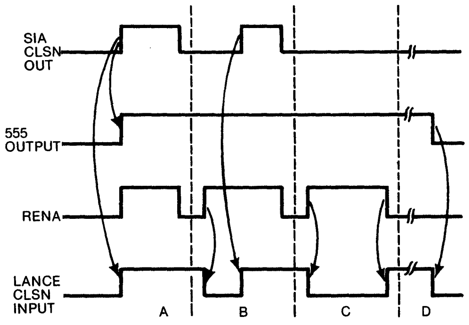


Fig 4—An example of interface signals *that occur when you use a priority circuit is shown here. In* **(a)**, *the SIA's Clsn signal goes High, turning the timer's output on. The LANCE's Clsn input is also driven High so it will neither receive nor transmit. A read-enable (Rena) signal* **(b)** *permits the LANCE to accept inputs, but a collision again puts the LANCE out of communication. In* **(c)**, *the LANCE is freed again to receive data until the Rena line drops, after which it resumes its quiet state because the 555 output is still High. When the 555 output drops* **(d)**, *the LANCE is free to transmit as well as receive.*

Author's biography

Vernon Coleman, as manager of local-area networking, directs the definition and logic design of semiconductor products for LANs at AMD Inc (Sunnyvale, CA). He's been with the firm for seven years and prior to that worked at Fairchild Semiconductor Co and Control Data Corp. He holds a BSEE from the University of California at Berkeley. His leisure-time pursuits include working out on weekends in his Oakland home.

Article Interest Quotient (Circle One)
High 482 Medium 483 Low 484

Protocol standardization works its way up the ladder of the OSI model

Using the seven-layer reference model for open-systems interconnection, standards organizations around the world are moving towards agreement

by David Berry, *Advanced Micro Devices Inc., Sunnyvale, Calif.*

☐ When the question of connecting heterogeneous nodes comes up, the International Organization for Standardization's seven-layer reference model for open-systems interconnection comes to mind immediately. But with so many groups and so many standards being developed, it is hard to discern the larger trends in network standardization. For the long term, open systems are a good bet, since encouraging signs suggest that the work is about to take off.

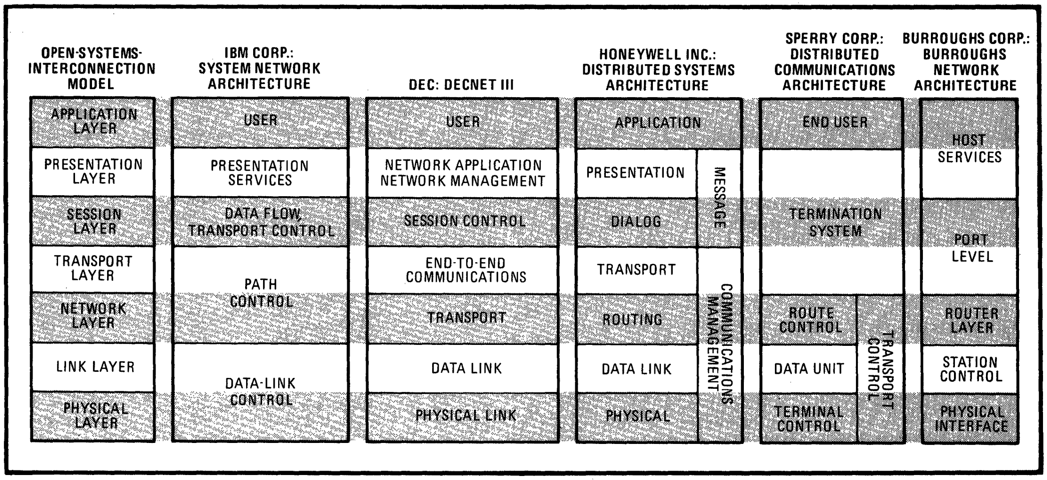
The granddaddy of all networks that link heterogeneous nodes is Arpanet, which the Defense Advanced Research Projects Agency started to develop in the late 1960s. This extensive network—linking mainframes in universities and Federal research laboratories across the U. S.—communicates through standard protocols that have defined much of the structure and many of the functions now regarded as essential ingredients of any open network.

Growing awareness of the potential of networks—demonstrated by the success of Arpanet—prompted standards-setting bodies to start guiding their development. In Europe, the European Computer Manufacturers' Association took a leading role. The fragmentation of the European computer market—the result of the Continent's tariff and language barriers—forced ECMA to become adept at securing compromise standards. Indeed, much of the ISO's work on standards is based upon earlier efforts by the ECMA—for example, the ISO's transport-layer protocol, which conforms to an earlier ECMA draft in everything but detail.

Lower-level concentration

In North America, much of the effort to develop standards has concentrated on media-access and lower-level protocols, though several committees of the American National Standards Institute are now developing higher-level ones. The most tangible result in the applications layer is the standard for videotex: the North American Presentation Level Protocol Standard (NAPLPS), adopted in draft on the international level. The National Bureau of Standards has actively issued specifications and organized user groups to work out practical implementations.

The OSI model has been the framework for all standards development. It is the map, giving users and manufacturers alike a path through the jungle of network functions. Currently, ISO has standardized network proto-



1. Match up. Although proprietary network protocols can be functionally aligned with one another and with the International Organization for Standardization's model for open-systems interconnection, direct compatibility among heterogeneous architectures still lies in the future.

cols all the way from the physical to the transport layer and is actively developing standards in the upper layers as well.

Mainframe and minicomputer manufacturers played a major role in the initial definition of the mechanisms required in networking. But they have not yet pushed for standardization, because up until now networks have been set up mostly in homogeneous systems environments where the problem of transferring data across equipment from different manufacturers was unimportant or resolvable by slow data-transmission mechanisms. Except for IBM's System Network Architecture, these proprietary networks will probably not have a large market impact outside their captive customer bases.

Although there is almost no direct compatibility among manufacturers' protocol structures, there is some similarity, and it is possible to make a rough alignment with the OSI model (Fig.1). Most of these manufacturers, including IBM, say they intend to provide OSI-compatible protocols for open systems, but how soon and how extensively they will do so is not yet clear. Certainly, their massive investment in proprietary protocols will not be scrapped overnight.

Another big player in the standards world is the International Consultative Committee for Telegraphy and Telephony, the regulating body of the telecommunications world, whose standards reflect its vested interest in the installed base of telecommunications systems. CCITT's tack will resemble the course that major manufacturers have taken with their proprietary networks, but its position carries much weight, and the S-series protocol standards from this organization have become an established part of all attempts to standardize layers.

State of the art

Most parties to the ongoing debate on standardization accept the partitioning of functions outlined in the seven-layer model, so the task of implementing protocols can be broken into more manageable components. A substantial measure of agreement exists on the details of the transport layer and on certain areas of the presentation layer, and as a result it is now possible to connect heterogeneous nodes on a local network and to have them pass data among themselves in a highly reliable manner.

The network layer is being defined, but the work has been hampered by disagreement on how the global network should function. Currently, the most common standard is the X.25 protocol used on public data nets, but this covers only the access sublayer and does not address the problems of harmonization or internetworking.

The session layer remains a battlefield. The ISO's spirit of compromise has produced a cumbersome proposed protocol that is largely unacceptable to U. S. interests as voiced by ANSI. As a result, the regulation of data transfer still is a matter of taste.

At higher levels of the model, the need for standardization increases, but the number of successfully completed documents is relatively small because of the more pressing needs of the lower layers and the degree of host dependence that functions acquire in the higher layers of the OSI model.

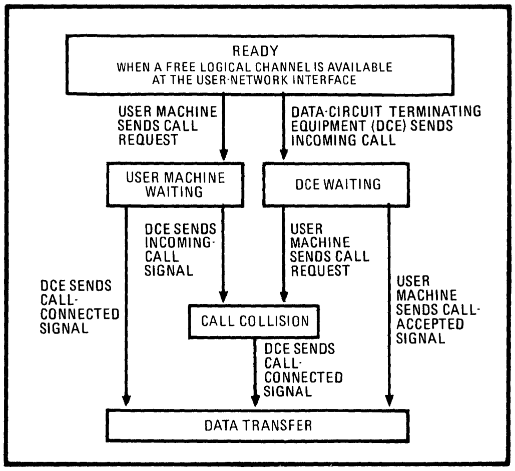
Packet-switched public data networks grew out of Arpanet and use the telephone system as their main network pathway. The protocols, standardized largely by the CCITT, are exemplifed by Canada's Datapac, Britain's EPSS, and the U. S. Telenet.

Packet switching improves the use of transmission paths between computers by permitting a high degree of multiplexing on the lines. Users can therefore vary the data rate to suit the terminal equipment they want to connect. Protocols complement packet switching, since packets provide fixed locations within the transmission, where protocol headers can be located and the frequent administrative messages can be sent.

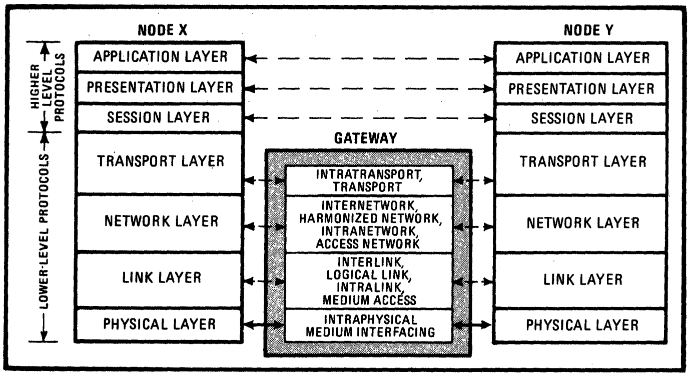
The most important protocol standard used in switching is the CCITT's X.25, which defines the relationship between the common-carrier packet-switched network and the user machines that employ it. The protocol makes no assumptions about the network's actual construction and is instead oriented towards connections available from common carriers.

The X.25 standard defines the interface between the host, or DTE (data terminal equipment), and the carrier, or DCE (data circuit-terminating equipment). A node that performs packet switching and forwarding functions is called a DSE (data switch exchange). The X.25 standard is concerned only with sending and routing data packets in a network and therefore defines only layers 1 through 3 of the OSI model. It is not concerned with the upper four layers and makes no attempt to define them.

In the X.25 protocol, the network layer is called the packet layer and provides routing and virtual-circuit management, which includes flow control and packet assembly and disassembly. The packet layer therefore includes several functions usually associated with the transport layer in the OSI model. While passing messages across the network, the entity that provides the protocol goes through several logical states (Fig. 2). Any idle virtual-circuit number can be used to designate the connection. The protocol has mechanisms to handle any


READY
WHEN A FREE LOGICAL CHANNEL IS AVAILABLE AT THE USER-NETWORK INTERFACE
USER MACHINE SENDS CALL REQUEST
DATA-CIRCUIT TERMINATING EQUIPMENT (DCE) SENDS INCOMING CALL
USER MACHINE WAITING
DCE WAITING
DCE SENDS INCOMING-CALL SIGNAL
USER MACHINE SENDS CALL REQUEST
DCE SENDS CALL-CONNECTED SIGNAL
CALL COLLISION
USER MACHINE SENDS CALL-ACCEPTED SIGNAL
DCE SENDS CALL-CONNECTED SIGNAL
DATA TRANSFER

2. Packet states. Packet switching across wide-area networks is a reliable way of interfacing heterogeneous networks. During the transfer operation, the protocol evolves though several logical states.



3. Gateway schemes. While gateways will probably be standardized at the network layer of the OSI model, it is possible to push down into the lower levels or up into the higher ones. Pushing the gateway upward eases the interface requirements.

conflict that call collision may create when these numbers are assigned.

X.25 uses a packet format that includes a number of fields in the header. Not all packet headers contain the same fields. A call request, for example, includes fields for the calling address, the called address, and the facilities required on the call, and these are not included once the virtual circuit incorporating them has been set up. Group and channel fields together specify the virtual circuit to which the packet belongs. Type and control fields classify the information in the packet. As a standard, X.25 has received support from major telecommunications users and has drawn several major manufacturers into the growing market for packet switches.

Wide-area-network upper-level protocols have been developed only by individual users, since packets pass between homogeneous remote systems. The packet network acts just as a neutral channel down which the data passes. Standardization, so far as it exists, has dealt solely with those protocols concerned with the actual transmission of data, but this will probably change with the introduction of common applications across company boundaries. Less sensitive services, such as electronic mail and graphics standards, are already leading the way. More sensitive areas, such as common data bases, must await the development of adequate security standards to protect company files from illegal access and abuse.

Local nets in front

But standardization in wide-area networks will take a back seat to developments in their local counterparts because local-net applications are going to explode. Wide-area networks will continue to handle much long-haul data traffic and may become dominant in backbone networks connecting local nets. If this happens, dedicated hardware devices for the physical layer and for parts of the link layer are likely to appear.

Standards for local-net protocols have benefited from the more mature standards developed for public data networks. But unlike public data nets, local nets have tended to embed more and more protocol functions in the interface between the node and the net. After the link-level functions have been provided in integrated circuits, attention will turn to integrating the software routines that provide higher-level protocol functions on chip.

This trend can be anticipated now because the typical local-net node is no longer simply the interface between an entire computer system and a complex network—as it is in Arpanet—nor even a way for a group of terminals to communicate with a remote computer through an X.3 packet assembly or disassembly. Instead, the node is becoming the link between the outside world and the data base of specific units, such as work stations, printers, and word processors.

However, the local-net market is still in its infancy, so silicon answers may be available in the future. Today's local nets are almost exclusively stand-alone systems where virtually all traffic originates and terminates internally. Up until recently, debate has centered around the technology of the network itself—particularly on the lower two layers, where the physical connection and its associated link-access protocols are located.

Functions associated with upper-level protocols are only now making themselves felt. The need for highly reliable data traffic has spurred on the standardization of the transport protocol. A series of embedded standards, up to and including layer 4, now provides the network market with a workhorse for data transfer between heterogeneous nodes. Routing and reliability have become transparent.

Gateways to the world

For the short term, at least, the means of transferring data between local nets from different vendors will remain nonstandardized, though it is possible to connect networks at any layer from the physical through the transport by creating a transparent connection or gateway. This is done at sublayer 3c—the internet protocol—in the current ISO transport-layer protocol. The degree to which protocols from one network must be translated to be useful in another one defines how the gateway will be implemented in practice.

The starting point for connecting networks has been the use of packet-switched wide-area networks as the common element, around which everything else is standardized. The isolation of current networks has generated a market for specialized gateway systems connected to packet-switched wide-area networks. In fact, the CCITT issued the X.75 recommendation to define gateways between local nets in general and packet-switched networks using the X.25 protocol.

X.75 and X.25 will probably continue to dominate the market for gateways. But a smaller segment of the market is the gateway between such proprietary nets as IBM's SNA and Xerox Corp.'s Xerox Networking System. This segment is currently important and will become more so relatively soon, a trend that will reverse itself as the market becomes dominated by standard networks. Because this market is fragmented, specialized systems will continue to provide solutions to the gateway problem.

The major difficulty in connecting wide- and local-area networks is in the difference in the types of network layers involved. LANs will probably use the connectionless protocol; wide-area networks, the more extensive connection-oriented network layer. One of the major requirements is to harmonize network sublayer 3b, where the two different kinds of services meet, so that the internet sublayer can route data between the two. Gateways will also be required for both connection-to-connection and connectionless-to-connectionless types of service, but this presents less of a problem and is not likely to occur as often.

Alternative gateway schemes, where the connection is made at such layers as transport, have been proposed, too. Connections at higher layers make it possible to mask the major incompatibilities of the connection and connectionless types of service networks. At lower layers, the degree of compatibility must be higher, but the gateway becomes correspondingly easier to implement. Figure 3 shows examples of several possible gateway schemes.

The future

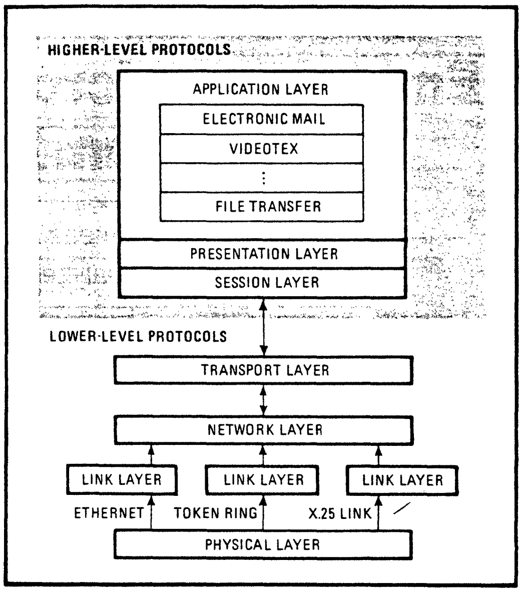
Above all else, the future network market will be characterized by diversity. Just as computers have become ubiquitous in everyday life, so too networks will become dispersed, pervasive, and focused on specific solutions. This is not to imply that they will be incompatible with the OSI model, but the division between the transport mechanism (layer 4 and below) and the transport user (layer 5 and above) will continue to mark different flavors of networks.

Network nodes will be connected through their transport mechanisms. A majority of the functions of layers 4 and below will be standard features of most nodes. Incompatible network technologies will create connections by using gateways. In descending order of importance, the major kinds of networks that will probably be available within five years are likely to be:

- Ethernet and compatible derivatives.
- The IBM token ring and clones.
- The token bus in industrial-control applications.
- Networks based on private automated branch exchanges, with the beginnings of an integrated services digital network.
- Packet-switching wide-area networks.
- Specialized miniature local nets interconnecting microcomputers.
- Proprietory mainframe nets.

The service provided at the boundary between layers 4 and 5 will probably become a universal standard. When this happens, the focus on network technology will fade and the focus on the applications run on the net will dominate.

In layers 5 and above, the protocols implemented in each node are likely to be application-specific—except for mainframes connected to the network, for they will be powerful enough to implement the complete range of protocols. But with the proliferation of intelligent terminals, work stations, and distributed data bases, the proportion of mainframe nodes will likely be quite small and decrease with time.



4. Upward to applications. As lower-level protocols mature, it will become easier to implement application protocols, which will probably proliferate as rapidly as application programs do now.

Application-specific nodes will run high-level user protocols that implement subsets of the full set of functions in the top three layers. The major subsets are likely to be videotex in home terminals; encryption in banking terminals; file transfer and distributed-data-base protocols in distributed systems; and electronic mail, videotex, and file transfer in desktop computers.

An indication of the diversity of the application protocols expected in the future is shown in Fig. 4, along with an indication of the major networks on which they will run. As a result of the market's size and potential, very large-scale integration will probably provide solutions for all major network technologies, particularly if none is dominant.

In the higher layers, the standardization that is expected to win out thanks to the widespread acceptance of the ISO's transport-protocol standard will ensure that an integrated solution is available. Higher than this, the degree of agreement drops alarmingly, and there is little likelihood that layers will be defined as clearly as the transport layer is. Since these functions are heavily oriented to software and the node operating system, additional lack of standardization here makes comprehensive hardware solutions unlikely.

What can be expected are a number of support devices that are to the node what floating-point chips are to a microprocessor. Likely candidates are the encryption, data-base-access, virtual-terminal, and file-transfer protocols. □

Sunil Joshi and Venkatraman Iyer, Advanced Micro Devices, Sunnyvale, Calif.

New standards for local networks push upper limits for lightwave data

Recent activities in ANSI committees foretell a different yardstick for high-speed local fiber networks.

While other standards bodies now specify top-end data rates of 10 Mbit/s for local networks, the American National Standards Institute (ANSI) is developing network standards for data rates an order of magnitude higher—around 100 Mbit/s. These are not futuristic speeds. Most companies involved in developing these standards plan to introduce products supporting such data rates during 1985 and are pushing to finalize the standard by mid-1984. Called the Fiber Distributed Data Interface (FDDI), this proposed new ANSI standard (currently in committee X3T9.5) specifies a token-passing ring architecture for local networks using optical fiber cable.

There are two reasons for the need for high data rate networks: a dramatic increase in computer processing power over the last few years and an enormous increase in the volume of stored or processed data. As a result, it is the lower-speed local networks that quickly become the weak link between devices needing to transfer huge amounts of data as quickly as possible.

Clearly, data rate requirements depend on the services supplied and the network's application. For example, back-end networks, which connect computers to other storage devices and peripherals, require high-speed data transfer. These networks have a fairly small number of nodes (frequently fewer than 50), and span relatively small distances—usually within a computer room. In general, a backbone network has to be at least as fast as the devices on it in order to minimize buffering constraints. As the speeds of hard disks and optical disks increase beyond 40 or 50 Mbit/s, back-end networks need to be even faster. In addition, protocols for such networks must provide for "streaming" operations, in which several data packets are sent end-to-end in a single network access. This is essential for disk transfers because, with streaming, disks can send entire tracks of data in one access, eliminating the wait for a complete disk revolution between network access opportunities.

Backbone

An organization wishing to connect its token bus, token ring and Ethernet networks would use a special bridge or gateway network to do so. Known as a backbone network, these interconnecting links can span distances of up to several kilometers, often connecting many floors of a building or even several buildings together. Since internetwork traffic is funneled through the backbone, it has to be able to accept fairly high data rates. Also, devices requiring high throughput can be connected directly to the backbone. Large mainframes in different buildings, for example, may have direct connections to the backbone. Or, a distributed PBX architecture can be implemented by connecting the local PBXs in each building via the backbone network, which likewise should be fast enough to carry real-time voice information.

Unlike back-ends, front-end networks typically connect computers to devices that are more user-interactive, such as terminals, word processors, workstations, and printers. Front-end networks can have hundreds of nodes spanning a few kilometers. The IEEE 802.3, 802.4, and 802.5 standards (see "Local network standards") are essentially designed for front-end applications.

Until now users have been satisfied with data rates of 10 Mbit/s or so for front-end networks. But if it were available—and affordable—most network users would welcome higher data rates and their services.

For example, a 10-Mbit/s data rate is not sufficient to support many real-time voice conversations, much

Local network standards

Numerous standards committees in the United States are standardizing various layers of the International Organization for Standardization (ISO) seven-layer Open System Interconnect (OSI) model. The most notable are the IEEE and ANSI committees, the primary focus of which has been ISO's physical and data link layers (table).

The IEEE is comprised of professional engineers, whereas ANSI membership is comprised of companies. ANSI is also the official body representing the United States in ISO. Hence, ANSI's organization closely parallels ISO's in terms of technical committees and subcommittees. Indeed, ANSI standards could eventually become ISO standards.

The IEEE defines standards for data rates of approximately 40 Mbit/s and below. ANSI focuses on data rates greater than 40 Mbit/s. Both committees have liaison members who represent them at the other committees' meetings—mainly to share expertise and to avoid duplication of effort.

The IEEE 802 committee is specifying a series of local network standards that define four access technologies for a variety of physical media. The four standards are described briefly as follows:

■ IEEE Standard 802.3. This is essentially the same as the Ethernet standard, which uses carrier sense multiple access with collision detect (CSMA/CD) as the access method. The standard is defined at a 10-Mbit/s data rate on a coaxial cable in a bus.

■ IEEE Standard 802.4. This bus standard uses a token-passing mechanism to determine network access. The primary application of this scheme is in factory automation where the advantages of bus topology are combined with the priority potential and deterministic properties of token access.

■ IEEE Standard 802.5. This defines a token-passing ring network. The medium is coaxial cable and the interconnection is a physical ring topology with a provision for passive bypass that uses relays.

■ IEEE Standard 802.6. This is a project, now under study, to put together a metropolitan area network.

ANSI activities

The ANSI X3T9 Committee has the charter to define I/O interfaces. Several subcommittees are working on standards.

■ ANSI X3T9.2 Small Computer Systems Interface (SCSI). Popularly called "scuzzy," this interface defines a scheme to interconnect low-end disk drives and other peripherals to computers.

■ ANSI X3T9.3 Intelligent Peripheral Interface (IPI). IPI defines a scheme to interconnect somewhat higher-end peripherals to host adapters over a parallel bus. At the link layer, IPI sends data in the form of packets, much like a local network.

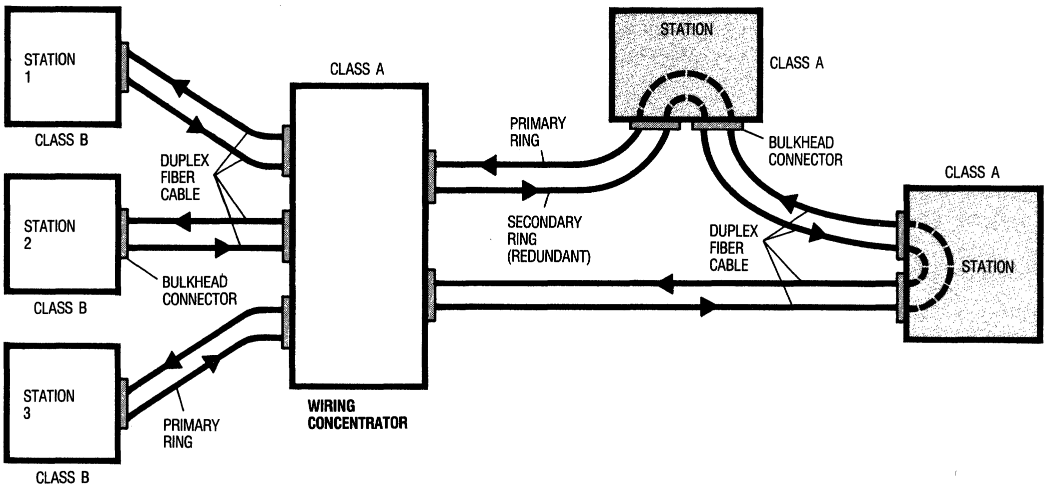
■ ANSI X3T9.5 Local Area Networks: Two subcommittees within X3T9.5 are defining different local network standards: Local Distributed Data Interface (LDDI) is in the process of modifying and adopting a 70-Mbit/s coaxial cable network proposed by Digital Equipment Corp. (DEC) based on their CI-network. It uses a star topology. And Fiber Distributed Data Interface (FDDI) defines a 100-Mbit/s fiber-optic ring, which uses a token-passing access scheme.

Standards activity in networking

APPLICATION	SC16	WG5	X3T5.5	—	—	—	ISO COMPATIBLE	—
PRESENTATION	SC16	WG5	X3T5.5	—	—	—	ISO COMPATIBLE	—
SESSION	SC16	WG6	X3T5.6	—	—	ECMA-75	ISO COMPATIBLE	—
TRANSPORT	SC16	WG6	X3T5.6	—	802	ECMA-72	ISO COMPATIBLE	—
NETWORK	SC6	WG2	X3S3.3	X.25	802	—	—	—
DATA LINK	SC6	WG1	X3S3.2 X3T9	X.21	802	ECMA-82	—	TR30.2 TR40.2
PHYSICAL	SC6	WG3	X3S3.1 X3T9	X.21	802	ECMA-81	—	TR30.1 TR40.1

TC	TECHNICAL COMMITTEE
SC	SUBCOMMITTEE
WG	WORKING GROUP
ISO	INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ANSI	AMERICAN NATIONAL STANDARDS INSTITUTE
ECMA	EUROPEAN COMPUTER MANUFACTURING ASSOCIATION
CCITT	CONSULTATIVE COMMITTEE FOR TELEPHONE & TELEGRAPH
IEEE	INSTITUTE OF ELECTRICAL & ELECTRONICS ENGINEERS
NBS	NATIONAL BUREAU OF STANDARDS
EIA	ELECTRONIC INDUSTRIES ASSOCIATION
TR	EIA SUBCOMMITTEE

1. Inner ring. *Primary and secondary fiber rings transmit lightwave data in opposite directions. Class A stations—either mainframes or wiring concentrators—connect to both the primary and secondary rings. Class B stations connect only to the primary ring. Fiber cable is terminated with bulkhead connectors.*



less video traffic. But as the need for these features increases with developments such as teleconferencing, higher data rates seem much more attractive. On a more practical note, low-speed networks are usually adequate for terminals and printers, but for engineering workstations, which require many huge file transfers daily, a high-speed front-end network is almost essential.

Designers have imagined many and varied fiber optic applications for years, but expense has kept many of those plans the stuff of dreams. Now, however, fiber costs are often comparable to those of coaxial cable and are expected to keep dropping. The breakthroughs have come from improvements in technology and the potential for mass production. Until recently, for example, it would have required a laser diode costing several hundred dollars to drive a fiber at over 100 MHz. This speed can be achieved now by using LEDs, priced at less than $10. In fact, one of the objectives of the ANSI FDDI specification group is to define the standard so that it can be implemented with fair-priced components within the reach of today's technology.

When cost is no longer a concern, fiber optics is a clear winner for a number of reasons. Four are:

■ *Bandwidth.* Fiber-optic cables allow high data rates—in the range of several hundred megabits per second.

■ *Attenuation.* Lightwaves experience relatively low signal loss, resulting in efficient communication over tens of kilometers, without repeaters.

■ *Noise susceptibility.* Fiber cables transmit information as light and neither generate nor are affected by electromagnetic interference (EMI). This makes it ideal for use in high EMI environments such as factory floors.

■ *Security.* Because it is difficult to tap a fiber cable without interrupting communication, fiber is more secure from interception.

The light-ring standard

As mentioned, the FDDI is a 100-Mbit/s token-passing physical ring using fiber-optic cable. The ANSI X3T9.5 committee specifies this as a local network standard. And various corporate architects of FDDI plan to use it for their own products because the standard includes features that support the many applications required for high marketability.

For example, some would use FDDI for back-end computer room applications where the network need only span several meters. Others intend to use FDDI-based products for connections with circumferences of over 100 kilometers. The FDDI specification places no lower bounds on the number of nodes and the distance between them; at the same time, the upper limits are reasonably large, permitting a wide range of implementations. Some of the main limits are:

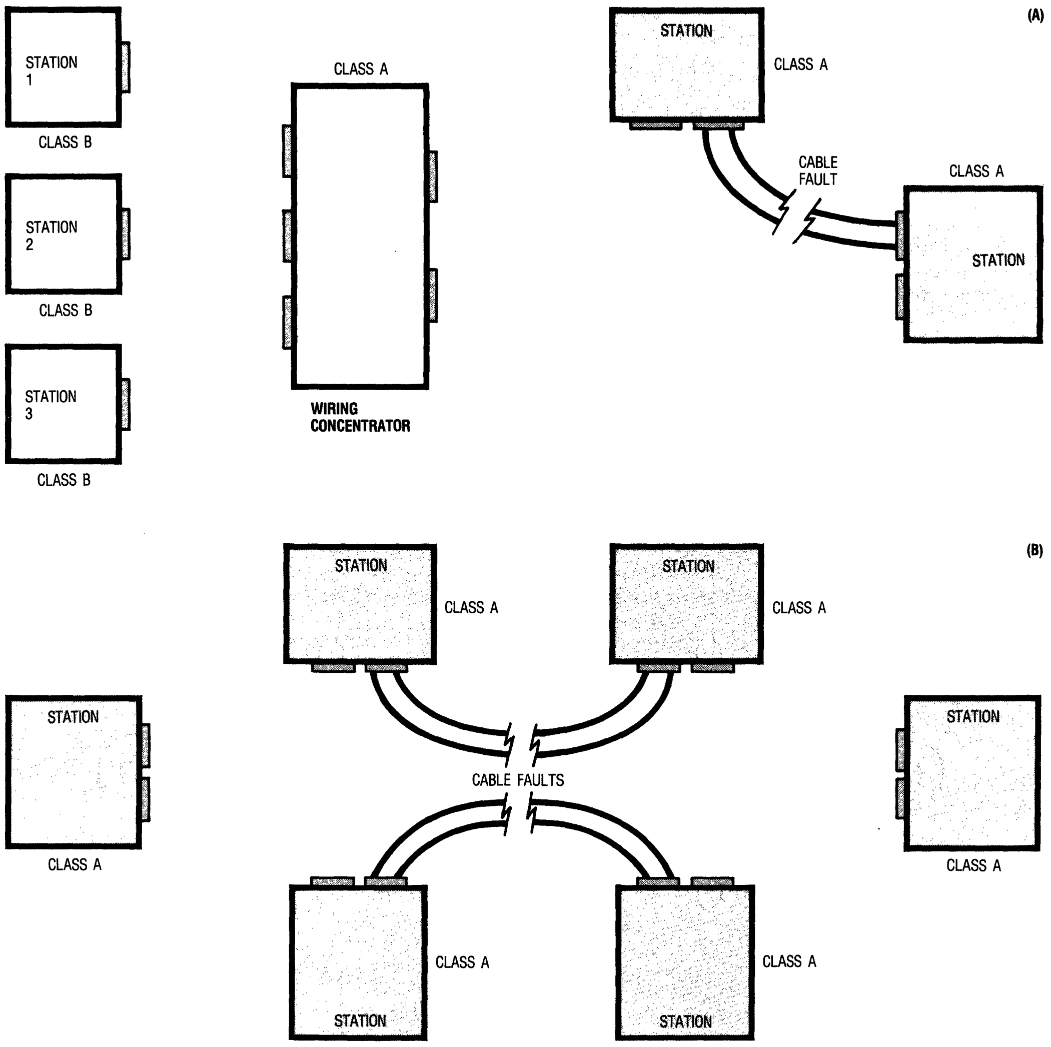
■ Up to 1,000 nodes on the ring
■ Up to 2 kilometers between two nodes
■ Up to 200 kilometers ring circumference.

With the maximum 1,000-node configuration, the average node separation will be 200 meters in order to limit the ring circumference to 200 kilometers. Yet several nodes may be separated by up to 2 kilometers as long as the average separation is 200 meters.

These limits are imposed to minimize latency, or the time it takes a signal to travel around the ring. The maximum ring latency is an important parameter for some real-time network applications. And with the proposed FDDI standard, it is held to only a few milliseconds.

continued

2. Fault tolerance. *As shown in A, an FDDI network can sustain a single fault and still maintain the integrity of the entire network. All stations simply reconfigure on the primary ring. If a network similar to that shown in B sustains two faults, it can split into two smaller rings which remain operative internally.*



The FDDI ring is a combination of two independent counter-rotating rings, each running at 100 Mbit/s. If both rings operate simultaneously, the effective throughput is 200 Mbit/s. An FDDI scheme can actually use a special case of this where one ring connects all the nodes, and the second counter-rotating ring only a selected few.

Figure 1 illustrates a possible FDDI network configuration with fiber-optic cables forming the inner and outer rings. The paths through which the data travels around the ring are also shown. The ring that reaches all of the nodes is called the secondary ring and carries data in the opposite direction of the primary ring. The primary ring connects only the class A stations (an explanation of station classes can be found below). Such a concentric scheme is useful during ring reconfiguration. If the outer ring fails, for instance, the network can continue operating on the inner ring and still keep the intact portions of the outer ring (Fig. 2a).

In an FDDI network, nodes or stations can be of two categories, either A or B (Fig. 1). Class A nodes connect to both the inner and outer rings; Class B nodes connect to only the outer one.

The two node classes allow users to tailor network complexity to meet cost objectives. For example, Class B stations need to connect to only one ring so they can be implemented for a lower cost. They are, however, isolated if that ring fails. On the other hand, while Class A stations may require additional hardware to connect to two rings, they offer a level of fail-safeness by being able to swap and continue operations on the good ring should the other fail. Typically stations that need more fault tolerance will be configured as Class A. Less critical stations can be Class B.

Another Class A station is the Wiring Concentrator (WC) illustrated in Figure 1. As its name implies, this device acts as a hub node through which several other stations can be connected. A WC allows a physical ring to be easily maintained, since, like a star network, failing nodes may easily disconnect. The WC can be a service point, with short connections taken out to individual stations, with several WCs so configured in a network. The WC is always a Class A station, with other Class A and B stations connecting to it.

An appealing package

The packaging of the fiber cable also makes the concept of WCs very attractive. As implied in Figure 1, the fiber cable in FDDI would contain two physical fibers packaged in one jacket with a bulkhead connector on either end. For example, for the dual ring connecting the Class A stations, each jacketed cable would contain one fiber for the primary ring and one for the secondary ring. For the Class B stations on the primary ring, the same cable would carry the incoming and outgoing data. Hence, when a Class B station is connected to a WC, only one physical cable need be routed between the two devices. Since the cable carries two fibers, a ring path is physically established.

Although implementing combined media, such as fiber and coaxial cable, is not specified in the FDDI proposal, it is easily accomplished. For example, a dual ring scheme connecting Class A stations can be built using fiber, but the shorter connections between a WC and the Class B stations can be coaxial cable. As long as the data rate is maintained and the token-ring protocol is adhered to, the existence of multiple media in a ring is transparent to the network.

The most typical faults in a network are caused either by failed components or broken cables. Even if a connection is not fully broken, there may be a substantial degradation that shows up as increased bit error rate. Figure 2A shows how an FDDI ring would reconfigure its data paths if a link—or a pair of links—in a dual ring cable became inoperative. The stations would sense the breakage and use the appropriate paths on the secondary ring to keep the network running. In FDDI this reconfiguration would happen automatically within a few milliseconds; the proposed standard defines a station management interface to facilitate this. Also, when a broken link is restored, the station management handshake will allow the ring to automatically return to its original state.

Figure 2B shows the effect of a second failure. In this case, the networks split into two smaller independent networks, both of which remain operative internally.

Nuts and bolts

If a fault occurs in the cable going to a Class B station, that station is cut off from the network and the WC bypasses the node. Situations in which stations connected to the ring lose electrical power or are turned off can be handled by optical bypasses. Electrically operated relays provide pathways that allow the fiber transmissions to bypass a node—so that when power is lost, a mirror directs the lightwaves through an alternative path. A bypass can be provided in either Class A or Class B stations. Most Class A stations that include WCs will probably implement bypasses to get the additional level of fault tolerance. A Class A node without an optical bypass will appear as if all its links are broken when powered off. Optical bypasses (or

4B/5B coding

Q	00000	QUIET
I	11111	IDLE
H	00100	HALT (FORCED BREAK)
J	11000	1ST OF SEQUENTIAL SD PAIR
K	10001	2ND OF SEQUENTIAL SD PAIR

		HEX	BINARY
0	11110	0	0000
1	01001	1	0001
2	10100	2	0010
3	10101	3	0011
4	01010	4	0100
5	01011	5	0101
6	01110	6	0110
7	01111	7	0111
8	10010	8	1000
9	10011	9	1001
A	10110	A	1010
B	10111	B	1011
C	11010	C	1100
D	11011	D	1101
E	11100	E	1110
F	11101	F	1111
T	01101	ENDING DELIMITER (USED TO TERMINATE DATA STEAM)	
R	00111	DENOTING LOGICAL ZERO (RESET)	
S	11001	DENOTING LOGICAL ONE (SET)	

V	00001	NOTE: THESE CODE PATTERNS ARE INVALID BECAUSE THEY ALLOW MORE THAN THREE CONSECUTIVE ZEROS IN A ROW.
V	00010	
V	00011	
V	00101	
V	00110	
V	01000	
V	01100	
V	10000	

even electrical bypasses) may be used in WCs at their Class B interfaces to mediate situations where the Class B stations fail.

The specification for FDDI physical layer includes the optical and mechanical characteristics of the fiber and the connectors, the power budgets, the dynamic range required in the receivers, and the encoding/-decoding scheme used.

FDDI specifies three different options for fiber: 100/400 micron diameter fiber (for example, 100 is the diameter of the fiber and 140 is the outer diameter of the cladding or jacket); 62.5/125 micron diameter fiber; and 85/125 micron diameter fiber. For all three, an 850-nanometer wavelength is specified. Bulkhead connectors specify the mechanical interface (Fig. 1).

Several of the low-speed standards, including the IEEE's, use Manchester encoding for baseband transmission. With this method, data and clock information are encoded together so each bit cell or bit period is divided in half—with line transitions occurring mid-way in each bit period. Each time a positive-going transition occurs, a "1" is sent; each time a negative-going transition occurs, a "0" is sent.

Unfortunately, Manchester encoding is only 50 percent efficient. Had it been used with FDDI at 100 Mbit/s, the encoding scheme would have required 200-Mbit/s bandwidth. Accordingly, the LEDs and PIN receivers would have had to operate at 200 MHz.

As a result, FDDI proposes a more efficient encoding scheme, called 4B/5B, to keep the baud rate down. In 4B/5B, the encoding is done on four bits at a time to create a five-cell "symbol" on the medium. The efficiency is 80 percent: to transmit 100 Mbit/s requires a bandwidth of only 125 MHz. The advantage is that inexpensive LEDs and PIN diode receivers, which operate at only 125 MHz, can be used.

The table shows the symbol encoding used in 4B/5B. It is an encoding scheme between 4-bit data "nibbles" and 5-baud symbols. The symbols are transmitted in a Non-Return to Zero Inverted (NRZI) line transmission format. The assignments between the four-bit nibbles and the five-cell symbols are chosen so that a transition is present at least once in three cells on the medium. Given an NRZI format, no more than three zeros could occur in a row, since, with NRZI, the absence of a line transition indicates a "0."

There are eight symbol combinations that violate this requirement, which should never occur in a properly operating ring. There are 32 possible code groups (or symbols) and 8 are invalid, so 24 are left for other interpretations. Sixteen of these correspond to the 16 data-values for a nibble. The remaining symbols are assigned special meaning as control characters at the physical layer. Some of the control symbols are used to derive line states such as "quiet," "idle," or "halt." Others are used as special delimiters for packets or fields within a packet. These include symbols denoted as J, K, T, R, and S. In FDDI these have been assigned specific meanings. For instance, J and K always occur in pairs and act as start delimiters for a packet (Fig. 4). The symbol T is used in conjunction with R and S as an ending delimiter.

Aside from delineating packets, these special symbols play a major role in station management handshake during ring reconfigurations. FDDI uses a token-passing ring consisting of stations serially connected by a transmission medium to form a closed loop. The packets are transmitted sequentially from one station to the next, where they are retimed and regenerated before being passed on to the "downstream" station. (Idle stations can either be bypassed or configured to function as active repeaters.) The addressed station copies the packet as it passes. Finally, the station that transmitted the packet strips it off the ring.

A station gains the right to transmit when it has the "token," which is a special packet that circulates on the ring behind the last transmitted packet. A station wanting to transmit captures the token, places its packet(s) on the ring, and then issues a new token, which the next station can capture for its transmission.

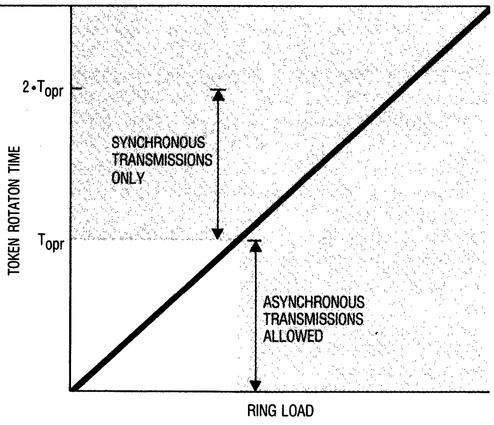
Taking turns

The access scheme in a network determines when a node has permission to transmit. The FDDI access scheme is based on a token packet which represents a permission to transmit. However, to support real-time needs of certain applications it is desirable to ensure that a node obtains a transmission opportunity (that is, assurance of getting the token) within a certain time.

The FDDI access scheme is called timed-token because it has a provision for specifying the time it takes the token to make a cycle of the ring. In essence each node on the ring measures the time it takes for the token to come back to it and compares that time with a prenegotiated target time. If the token comes back sooner than the target time, there is probably a light transmission load on the network. The node can then initiate packet transmissions on the ring as long as it does not exceed the target time. When the token comes back later than the target time, a heavy load on the ring is implied. In such cases the node is only allowed to transmit a high-priority class of packets called "synchronous packets." The node has to wait until the token rotation time (TRT) gets shorter than the target time before it transmits any of its more mundane packets that are not synchronous.

In addition, the FDDI uses the timed-token scheme to allocate the ring's bandwidth. This scheme assumes that the TRT, or the time between two successive token sightings at a station, has a linear relationship to the load on the ring (Fig. 3). The operative TRT T_{opr} is the same at all nodes on the ring. It is guaranteed in a normally functioning ring that TRT will never exceed 2 T_{opr}. The TRT would normally be less than or equal to T_{opr} transient conditions it may exceed it. Hence, nodes that have synchronous traffic (voice, for example) are assured at least one transmission opportunity in every 2.T_{opr} period. This is an upper bound on TRT; the TRT itself may vary from one revolution to the next. Thus, synchronous stations need some minimal buffering to support their traffic. The node that needs the lowest TRT determines the T_{opr} for the entire ring through a bandwidth allocation algorithm to be specified. The access scheme is independent of data rate and pro-

3. Upper limits. *TRT is a direct function of loading.* T_{opr} *is a prenegotiated length of time in which each node normally expects the token to return.*



T_{opr} = OPERATIVE TOKEN ROTATION TIME, A PRENEGOTIATED EXPECTED TIME BETWEEN TOKEN SIGHTINGS ON THE RING.

vides an easy transition to higher performance rings. The total synchronous load is the sum of synchronous traffic at the synchronous nodes. Thus, the maximum synchronous load that can be supported, as shown in Figure 3, sets up the configuration limits on the ring.

A TRT less than T_{opr} implies that the load on the ring is light. Asynchronous traffic (file transfers, keyboard commands) can be supported and prioritized at such times. Low-priority packets can be transmitted only when the ring load is light—that is, when TRT is less than a certain threshold for that priority level. In this fashion, the timed-token scheme provides a simple but powerful way to support synchronous and asynchronous traffic on the same ring, with an additional capability of dynamically prioritizing asynchronous traffic as a function of ring load.

Error recovery

Ring malfunctions such as noise, babbling nodes, or configuration violations are reflected in the TRT. And, if the TRT exceeds $2T_{opr}$, then ring malfunction is implied. (FDDI also specifies an early warning timer which times out before $2T_{opr}$ and initiates recovery.) All the nodes then act in concert to recover the ring. Each starts transmitting "Claim Token" packets that contain the TRT value needed by that particular node to support its synchronous traffic. The node with the lowest TRT wins. However, in the case of a broken ring, this process does not go to completion as indicated by a TRT exceeding a threshold value. The nodes start sending out beacon packets, indicating serious ring failure, and any node that receives a beacon repeats it. This enables the fault to be localized and triggers the reconfiguration procedure, thus enabling the ring to function again—albeit at reduced performance. The FDDI standard also specifies the handshake procedures for such things as a station that is either getting onto a running ring or going to a preconfigured setup.

Figure 4 shows the format for an FDDI packet; the token is a special packet and is also shown.

The packet preamble (PA) consists of at least idle symbols. The start delimiter (SD) consists of the symbols J,K (table). The frame control (FC) is 1 byte, and indicates frame type. The destination/source address (DA/SA) can be either 16 or 48 bits long. The address length is specified in the FC field. The information field (INFO) can have from 0 to approximately 9,000 data symbols. An autodin cyclic redundancy check (CRC) comprises the frame check sequence (FCS). End-of-frame (EDF) is followed by frame status (FS), which is provided to the transmitting node by the nodes through which the packets pass.

Comparing

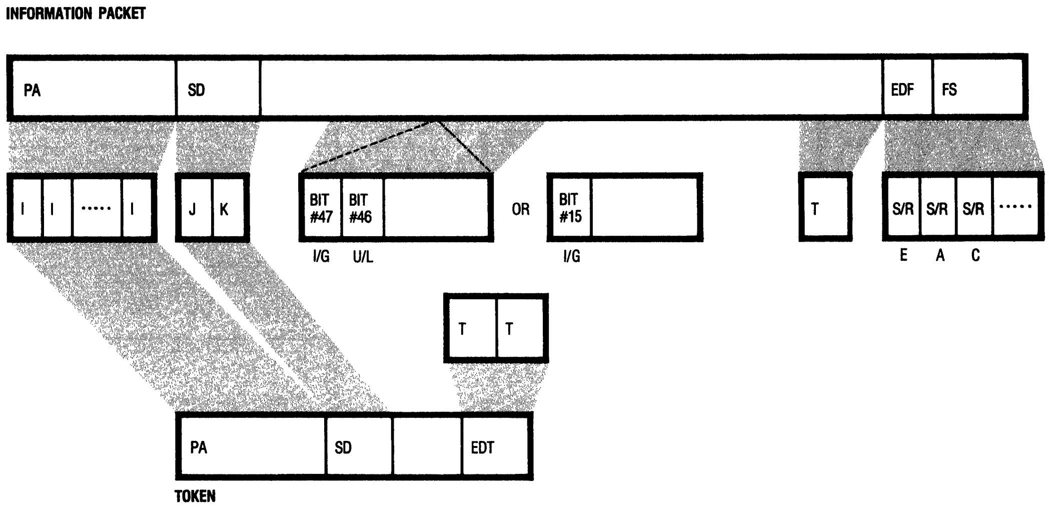
The IEEE 802.5 standard specifies a 4-Mbit/s token-passing ring. When the FDDI subcommittee began work it tried to maintain the services and facilities of 802.5 but modified other sections to obtain data rates up to 100 Mbit/s. A comparison between the two standards will illustrate the factors to be considered when going to higher data rate protocols. Specifically, the physical layer specification is crucial at high speeds. For a data rate of 100 Mbit/s, the group encoding limits the bandwidth to 125 MHz on the medium; as mentioned, a Manchester encoding scheme would have required an increased bandwidth of up to 200 MHz for the same 100-Mbit/s data rate.

FDDI's decentralized clocking scheme requires designing for clock variations between two stations instead of designing for the sum of clock variations on the whole ring. At high speeds, where the bit time is very small, the latter scheme would put tough (and therefore expensive) requirements on the clocking circuitry. Both schemes require latency buffers. However, the buffer in FDDI is divided among the nodes. In IEEE 802.5 each node has a buffer large enough to compensate for all clock variations on the ring. At high speeds the number of bits of buffer goes up and so the 802.5 scheme becomes impractical. The 802.5 centralized clock scheme supports packets of any length. FDDI packets are limited to around 4,500 bytes.

In the 802.5 standard a transmitting node flips a bit in the token—making the bit a start delimiter—and then appends its own packet behind the token. This means that a packet has to be ready and set up for transmission when the token appears. This also means increased hardware and costs. In FDDI the token is a special packet that is stripped off the ring by the node that wants to transmit. When a node takes the token, it puts idles on the ring while the packet is getting set up for transmission. Symbol level manipulation is sufficient in FDDI, unlike in 802.5 where bit-level manipulation is necessary.

From the above it appears that the IEEE scheme is more efficient. However, at high speeds it is the ease of processing, not the efficiency of data transmission, that causes the bottleneck. To ease the speed, size, and complexity requirements on the electronics, the price of a few bytes is an easy tradeoff. In FDDI the token is

4. Data bars. *Information is passed around the ring in frames. The PA field synchronizes the frame with the node's clock. Other fields indicate a frame's beginning, length, destination, source, contents, data integrity, and end. The token is similar to an information packet, but with no information field.*



PA	= PREAMBLE	FS	= FRAME STATUS
SD	= START DELIMITER	I/G	= INDIVIDUAL/GROUP ADDRESS
FC	= FRAME CONTROL	U/L	= UNIVERSAL/LOCAL ADMINISTRATION OF ADDRESS
DA	= DESTINATION ADDRESS	E	= ERROR DETECTED
SA	= SOURCE ADDRESS	A	= ADDRESS RECOGNIZED
INFO	= INFORMATION	C	= COPIED
FCS	= FRAME CHECK SEQUENCE		= DATA SYMBOLS
EDF	= ENDING DELIMITER (INFORMATION)		= SPECIAL SYMBOLS (I, J, K, T, R, S; SEE TABLE 1)
EDT	= ENDING DELIMITER (TOKEN)		

transmitted immediately after the packet. The 802.5 configured ring waits for the transmitted packet to come back before giving out the token, putting out fill (null data) in the meanwhile. The FDDI scheme is thus more efficient, especially in large rings with a lot of latency.

The FDDI timed-token scheme sets up a traffic framework through a bandwidth allocation scheme. This framework is maintained until some node wants it changed. Synchronous traffic is easy to support on the FDDI ring, and the TRT is the single parameter that controls the ring. In 802.5 each frame has three priority bits (to indicate one of eight levels of priority) and three reservation bits to indicate the priority desired. By suitable manipulation of these fields, synchronous traffic can be supported. Note, however, that traffic is regulated on a per-packet basis.

The dual-ring option specified in FDDI provides the additional reliability and fail-safeness essential for high-performance applications. Because the 802.5 standard is aimed at the low-end market, the dual ring is not specified, though it could be supported.

The success of FDDI will depend on the amount of support it receives from component manufacturers, and the cost reductions achieved as users and manufacturers progress along the curve involved with learning how to implement the standard.

Toward that end, Advanced Micro Devices (AMD) plans to introduce the first chip in its Supernet family of high-speed networking products in mid-1985. The first chip in the family, the Link Protocol Processor (LPP), does most of the buffer management functions, and the FDDI ring-access scheme can be implemented as a module to work with it. ■

Sunil Joshi, section manager for AMD's packet networks division, is involved in product planning for high speed local networks. Venkatraman Iyer is a product planning engineer for that division.

VLSI and digital networking

Confused by terms like ISO, X.25, Ethernet, ISDN? Read on for an in-depth look at cable and fiber optic waveguide systems, access methods, and VLSI circuit solutions.

Vernon Coleman
Advanced Micro Devices
Sunnyvale, California

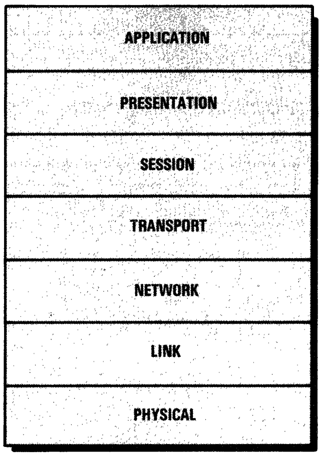


Fig. 1. Most LANs follow a "layering" scheme called the International Standard Organization open system interconnect (ISO OSI).

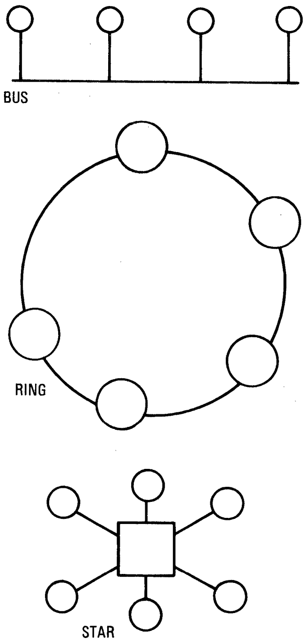


Fig. 2. Three basic architectures or topologies define a LAN. They are bus, ring, and star.

A local area network (LAN) is a means of moving data over short distances between a network of receivers and transmitters. You, as a designer of such networks, must consider a maze of facts in order to successfully create an optimal information transport system. Speed, architectures, and the ease of implementing the network from a hardware viewpoint must be examined in detail.

Most LANs follow a layering scheme called the International Standard Organization (ISO) Open System Interconnect (OSI) model—ISO for short. The ISO model is composed of seven layers (see Figure 1).

Each layer perfoms a distinct function in the LAN scheme. The physical layer is first and is concerned with electrical characteristics, physical connection means, and line encoding.

Next comes the link layer, which specifies how the ones and zeroes passed to it by the physical layer are interpreted. The link layer is responsible for address detection and making sure that the data delivered across the local medium is error-free.

Above the link layer resides the network layer. In general, this layer takes care of moving messages from one LAN to another. The complexity of the network layer depends on whether the messages are being routed to a similar LAN or a dissimilar one.

The delivery of error-free data is the fourth, or transport layer's responsibility. Messages arrive at the nodes in discrete chunks called packets. A datum delivered across the local area network may be error-free at the bit boundary, but the packets themselves may be out of order. This happens in networks that have multiple paths to given nodes. The transport layer will make sure that the datum sequence is correct.

Above the transport layer is the session layer. The session layer's function is to establish and terminate connections between nodes, mapping node names to physical node addresses, and generally controlling the length of messages that will be passed between the communicating entities.

The syntax of transmitted and received data is taken care of in the presentation layer. It acts as a language translator, allowing nodes to communicate using the same character sets, text strings, data display formats, etc.

Riding on all of this is the applications layer, the seventh and final portion of the ISO model. Such things as password checks, file transfer, videotex, and banking protocols are managed here.

Let's Get Physical

One of three architectures can define a local area network at the physical layer of the ISO model. These are the bus, the ring, and the star. See Figure 2.

The bus, like other architectures, consists of a one-bit-wide medium that carries both clock and data. In the bus scheme all the residents on the network are electrically connected by the same physical medium. The devices that reside on the bus usually have intelligence, which will allow them to access the bus without a central controller.

These intelligent nodes either gain the bus by sensing no transmission on it, or are given per-

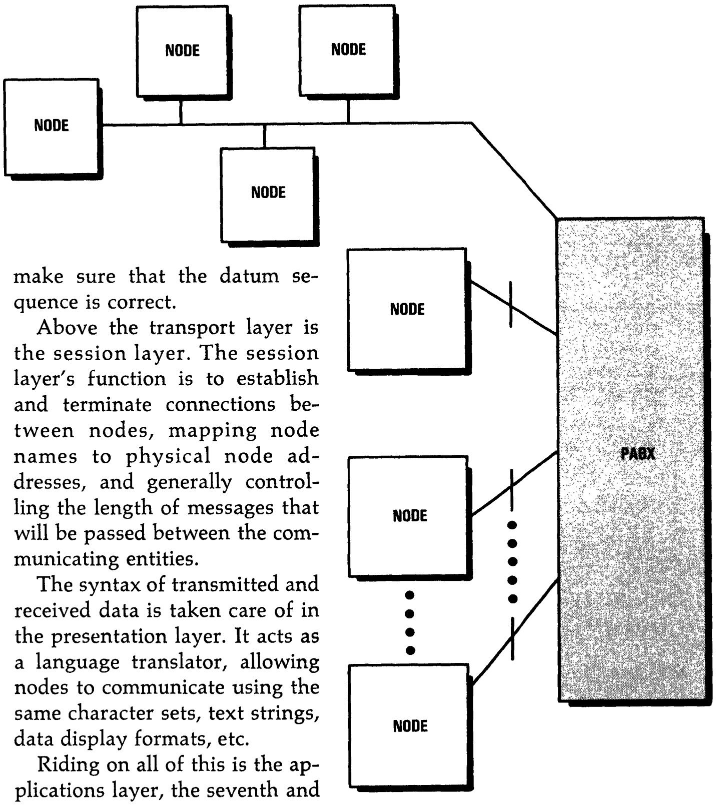


Fig. 3. The Integrated Services Digital Network, otherwise known as ISDN, is a LAN scheme providing an interface between multiple nodes and a PABX. Up to eight devices can be connected to each port.

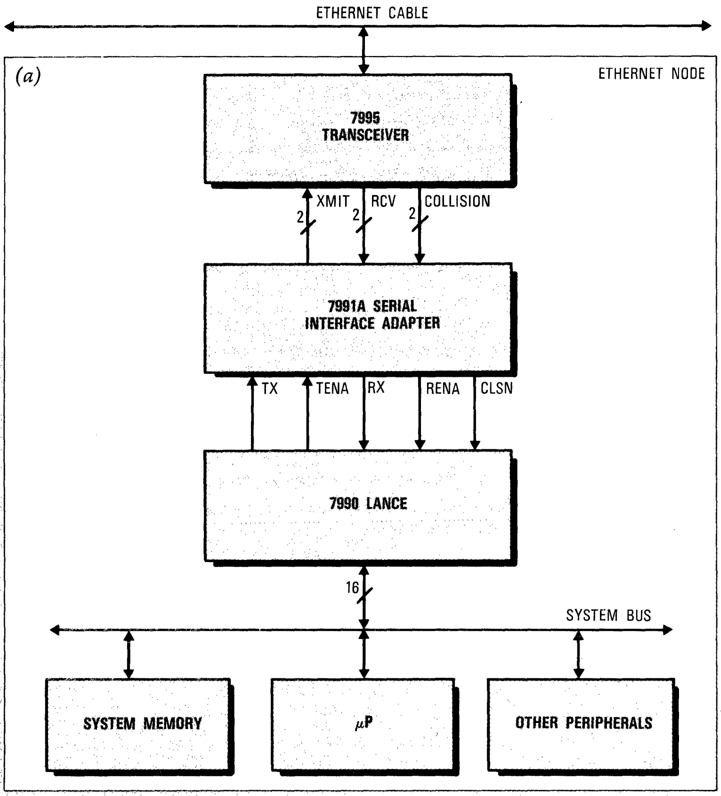


Fig. (a). In this node, three specialized VLSI devices are used to connect a node to the network. These ICs are an Ethernet LAN controller, a serial interface adaptor, and an Ethernet transceiver.

SOME SILICON SOLUTIONS

Once an architecture is chosen, the next step is to find the optimal ICs. Solutions can be found in a range of VLSI circuits for LANs from a variety of vendors. Here are examples based on future products presently under development at Advanced Micro Devices.

For the star configuration of ISDN, for example, AMD is readying four ICs that will allow connection and processing of digital information between nodes and a PABX.

These ICs will control the 'D' channel and two 'B' channels for an aggregate data rate of 144 Kbits per second. The family's functionality provides the flags, zero insertion/ deletion, and CRC generation/ detection according to the Link Access Protocol D (LAPD) format as well as resolving any resulting bus contention on the 'D channel.

Also, the chip set for ISDN provides for a full range of features to allow easy connection to a voice handset. Other functions will include power control, and the ability to interface the B and D channels to the PABX.

ICs for the bus structure of Ethernet/IEEE 802.3 are the Am7990 local-area-network controller for Ethernet (LANCE), the Am7991A serial-interface adapter (SIA), and the AM7995 Ethernet transceiver. Typically, these three chips connect a node to a network as shown in Figure (a). A discussion of other VLSI approaches to packet generation and transmission was covered in a recent article in **Integrated Circuits Magazine**, *October, 1984, page 39.*

The 7990 LANCE is the primary link between the node's central processor and the Ethernet system. It enforces the CSMA/CD access protocol, manages external memory for transmission and reception, and reports errors.

The 7991 SIA encodes and decodes data going to and coming from the main coaxial cable using Manchester encoding and decoding. It also converts the TTL I/O of the LANCE to the differential I/O of the transceiver. For incoming data, the SIA recovers the clock with an on-board phase-locked loop. For outgoing data, either an external crystal oscillator or timing inputs provide the clock.

The 7995 transceiver complies with Ethernet and IEEE 802.3 specifications, but it will also offer the option of operating in networks with longer electrical lengths. It will contain jabber-control circuitry as well.

To transmit a packet, the LANCE initiates a DMA cycle

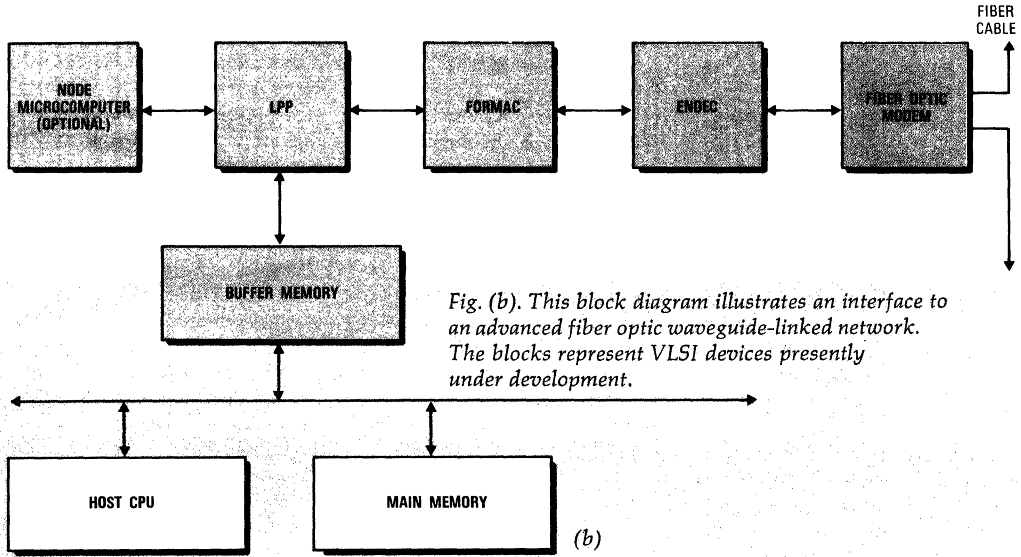


Fig. (b). This block diagram illustrates an interface to an advanced fiber optic waveguide-linked network. The blocks represent VLSI devices presently under development.

and pulls data from a transmit buffer set up in system memory. It frames the data by prefacing it with a preamble and sync pattern and appending a 32-bit cyclic redundancy check (CRC), which the LANCE calculates.

It transmits the packet with CRC serially to the SIA, which in turn creates the Manchester-encoded differential signals to drive the transceiver. The transceiver then converts this signal to a single-ended drive on the coaxial cable.

When it receives a packet, the transceiver creates differential output signals to drive the SIA's inputs. These inputs to the SIA are decoded by its Manchester decoder. Its phase-locked loop synchronizes to the incoming packet's preamble, allowing the decoder to recover clock and data from the encoded signals.

In addition, the SIA creates a receive-enable signal while it is receiving, indicating to the LANCE that received data and clock signals are available. When the packet reaches the LANCE, the preamble is stripped and the CRC is checked for correctness. The packet and its associated status are automatically stored in receive buffers in main memory.

The interface to the FDDI ring can be accomplished with three VLSI devices, now in development at AMD, and an optical fiber modem. See Figure (b).

The data to and from the modem is passed through an Encoder/Decoder (ENDEC). The ENDEC's function is to code and decode clock/data using FDDI's 4B5B run-length-limited line encoding format. The unencoded data going to and coming from the ENDEC passes through a Fiber Optic Ring Media Access Controller (FORMAC) chip.

The FORMAC is responsible for the protocol that governs nodal access to the medium. Such functions as token handling, synchronous and asynchronous data priorities, as well as immediate network recovery priorities are managed in the FORMAC.

In case of network failure the FORMAC, in conjunction with the ENDEC, will initiate recovery and network node notification procedures.

The FORMAC removes access data from the incoming information stream and sends it to a Link Protocol Processor (LPP). The LPP is a highly flexible programmable link controller which can accommodate user-defined protocols.

The LPP also has arbitration logic to alleviate contention for the tightly coupled memory which it manages. A microprocessor interface is also featured in order to allow optional pre- and post-processing of incoming or outgoing data. Other features include a complete set of status and error reporting bits.

mission by another node to transmit. Sensing lack of transmission is called carrier sense multiple access (CSMA) and is the basis for such popular bus protocols as the IEEE 802.3 standard, otherwise popularly known as the Ethernet.

A bus that passes permission to transmit between nodes is called a token-passing bus (the token being the permission to transmit). The token-passing bus is described under IEEE 802.4, and has been mainly adopted for factory automation.

Token passing is also used in ring networks. Rings must pass data sequentially through all the nodes in the network, therefore making token passing the natural access method. Rings are used in LAN schemes such as the IEEE 802.5/IBM and the American National Standard Institutes (ANSI) Fiber Distributed Data Interface (FDDI).

Networks that connect all the nodes to a central control point are called stars. A typical star network would be found in a PBX. In actual practice, a CSMA and ring architecture may be the central control point with nodes radiating from it.

What Are the Tradeoffs?

It is useless to speak only of the architectures themselves—they must be related to real implementations. A network designer must look at the emerging standards, which not only mandate specific architectures but dictate access methods and bit rates as well.

The LAN architecture that has been in existence longest is the star. The Private Automatic Branch Exchange (PABX), which is a star, is the common means for distributing voice within a campus environment, for example. Although the majority of PABXs are presently using analog interfaces, the trend is toward digital.

Enter ISDN

The premiere standard toward which the PABX is migrating is the Integrated Services Digital Network (ISDN). ISDN is actually more than a LAN scheme since it provides for interfacing the PABX to central offices as well. However, we will discuss here only the terminal-to-PABX interface.

ISDN is a four-wire interface which can connect up to eight devices on a single port to the PABX (see Figure 3). Each ISDN port of the PABX can accommodate two 64 Kbit per second voice or data circuit switched channels, one 16 Kbit per second control channel, plus 48 Kbits per second for framing and overhead.

The lines into the PABX can be up to 150 meters long in a widely spaced shared bus configuration or up to 1000 meters for a single station in a point-to-point connection. Furthermore, ISDN will allow up to 1000 meters for a single station in a point-to-point connection, or a concentrated group of stations in a point-to-multipoint configuration. ISDN allocates channel bandwidth between the node and the PABX through time division multiple access (TDMA). The 16 Kbit per second 'D' subchannel utilizes Link Access Protocol D (LAPD) to communicate control and packet data. Two 64 Kbit per second 'B' subchannels are simply pipes that have no associated protocol.

A strong engineering advantage of ISDN and true star (point-to-point) architectures in general is that there is a dedicated set of wires for each piece of equipment attached to the network. This guarantees that each node may have up to 128 Kbits per second of bandwidth.

In addition, hundreds or even thousands of nodes may operate simultaneously. Thus the aggregate throughput of an ISDN network can be in the hundreds of megabits per second. A plus for the system is its ability to handle real-time voice as well as data. In most digital systems real-time voice will occupy at least 64 Kbits per second of bandwidth per node. This requirement restricts real-time voice usage in other networks.

One drawback of ISDN, as of most star architectures, is the data rate capacity of the PABX central hub. Although ISDN can provide up to 144 Kbits per second to a node, this is still not a data rate sufficient to handle certain classes of peripheral devices that may reside on the network.

One example is the sharing of the ubiquitous 5.25-inch hard disk, whose transfer rate is 5 megabits per second. This means that the data from the disk would have to be buffered before being sent to the requesting node. The overall effect is to add expense to the interface, while also increasing response time.

One other concern with star architectures that contain active hubs is the central point of failure. Although the PABX is highly reliable as a central hub, in some applications such as manufacturing and process control, this reliability is not considered enough. For those applications which require higher transmission rates and/or are concerned with central points of failure, a bus architecture may be optimal.

Ethernet Comes First

The first and best example of a working bus architecture is

Ethernet and its near duplicate, the IEEE 802.3. These bus specifications allow up to 100 nodes to be attached to an active cable segment of 500 meters.

The cable segments themselves can be joined by repeaters to form a flexible network layout with up to 1024 nodes (see Figure 4). The Ethernet/IEEE 802.3 standard mandates a 10 megabit per second data rate. The nodes on the network arbitrate among themselves for transmission rights by employing Carrier Sense Multiple Access with Collision Detection (CSMA/CD).

In this form of CSMA, after sensing the lack of carrier on the media the node is free to transmit. In the case where two or more nodes begin transmitting simultaneously, the data on the media will collide. The transmitting nodes sense the collision and cease transmitting (or back off) for a random amount of time determined by each node.

Try, Try Again

After the node's backoff time is complete, it again checks for carrier before attempting to transmit again. The node may attempt retransmission for a single packet up to sixteen times before flagging an error.

The benefit of the Ethernet/IEEE 802.3 architecture is its simplicity of implementation, reliability, connectability, and data transport capability. In the bus system, unlike in the star, there is no central hub, since the intelligence for network access and arbitration is distributed. Networks may be implemented on a small scale initially without the capital outlay of a central hub such as a PABX. The ultimate number of nodes on the network is not bound by the capacity of the central controller, but in theory may be expanded infinitely by the use of network repeaters.

The lack of a central point of failure makes bus networking highly reliable. The Ethernet/IEEE 802.3 specifications, for instance, have provisions to automatically disconnect a faulty node from the network, therefore allowing continued system operation. Installation of new nodes on a bus is very simple. Ethernet/IEEE 802.3 allows a clamp-on connection, which can even be made while the network is in operation.

Ever Faster

An important attribute of architectures that share a common medium is high data rates. This allows the many potential contenders for the medium to gain access and transmit their messages in a minimal amount of time. Higher data rates also ease interface requirements for high performance peripherals to the network. The 10 Mbit per second data rate of Ethernet/IEEE 802.3 allows such devices as the 5.25-inch disk to be networked economically and with no performance disadvantages.

For the most part, however, bus implementations are more difficult to engineer because of the increased effect of impedance mismatches as the bit rate of the network increases. Although there are commercial implementations of bus networks that run in the several tens of MHz, the connections to these networks are prohibitively expensive with very high-cost nodes.

In addition to purely physical limitations at ultra high speeds, there is also a decrease in the efficiency of the CSMA/CD protocol, which the majority of local area bus networks employ.

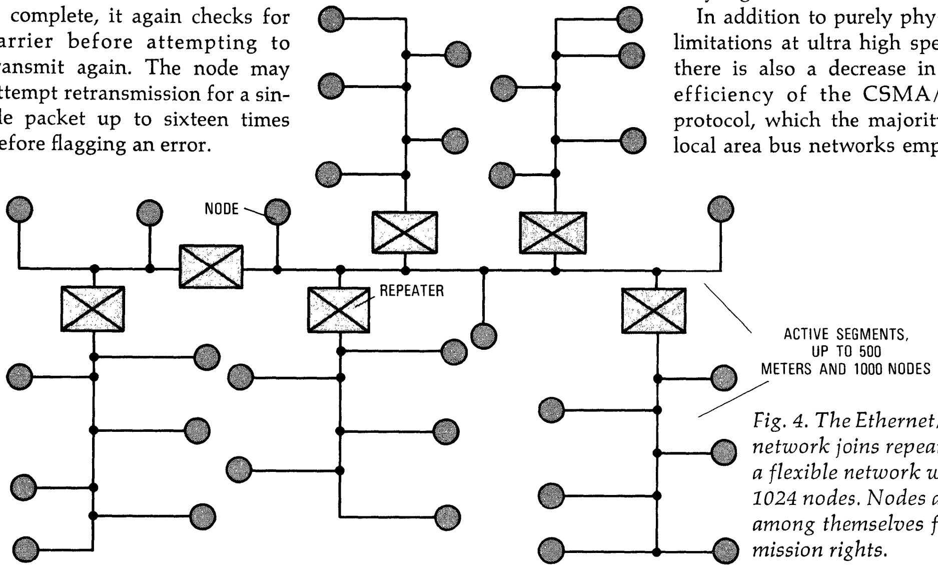


Fig. 4. The Ethernet/IEEE 802.3 network joins repeaters to form a flexible network with up to 1024 nodes. Nodes arbitrate among themselves for transmission rights.

For any fixed electrical length, number of nodes and bus loading parameters, the data rate is inversely proportional to the network efficiency.

Efficiency is defined as the time the network spends transmitting valid data versus the time it spends contending for transmission. For a 100 per cent loaded bus, efficiency (E) is calculated by

$$E = \frac{L/R}{L/R + ws}$$

where R is the bit rate, w is the mean waiting time to transmit a message, and s is twice the signal propagation time between the farthest points of the network. L is the length in bits of the message being transmitted.

To ascertain w, you must first calculate the probability P of only one of N stations beginning a transmission in the discrete time period s. Then, 1/N is the probability that a given node will begin transmitting in the time period s, and is the probability that no other nodes will begin transmitting in the same period. Therefore,

$$P = N(1/N)(1 - 1/N)^{N-1}$$

This equation can be simplified to

$$P = (1 - 1/N)^{N-1}$$

The mean value of this probability distribution is

$$w = (1 - P)/P.$$

You can now readily see the effect of the various parameters on efficiency, E.

For about twenty or more stations, the value of w approaches 1.7. If we want more efficiency from the network at higher speeds we must either increase the packet length, decrease the length of the bus, or settle for a combination of the two.

It should be noted that overall throughput is increased by transmitting at higher speeds. However, for most network implementers, the inherent waste due to inefficiency is not acceptable.

As an example, if we are transmitting maximum-length Ethernet/IEEE 802.3 packets of 12,144 bits on a standard network where s is always mandated to be 51.2 microseconds, at a data rate of 10 Mbits per second, then, by the derivation given, E equals 0.933. This means that on the average, the network has a true throughput of 9.33 Mbits per second.

Throwing Away Speed

If we raise the data rate to, say, 100 Mbits per second, the network will yield a value of E that equals 0.582 or 58.2 Mbits per second. What an implementer will have done is engineer a 100 Mbit per second network that under heavy loading will carry much less data.

The electrical parameters of the bus become an increasing source of data errors at very high speeds. At higher speeds the mismatch in impedance between the stub of a nodal tap and the transmission medium must be controlled very tightly otherwise electrical reflections and data errors will result.

Another problem for buses at very high speeds is that the signal is distorted by cable attenuation at higher frequencies. This cannot be easily compensated for unless expensive radio frequency technology, such as broadband modems, are used.

There are also some applications, such as factory and process control, that must guarantee network access to a particular node in the system. Access to the network is probabilistic in CSMA systems.

Guaranteed access ranks just behind reliability in importance in most factory environments because reading status from, and sending commands to, controllers in many cases has to be done at predictable intervals. To obtain predictability, a more deterministic access scheme must be used.

The solution to the very high speed bus problem may be found in the token-passing ring scheme. Token passing in itself eliminates the occurrence of collisions, therefore making data transmission more efficient over shared media at higher data rates. Token passing is also deterministic, making the scheme more suitable for real-time applications.

Rings offer two distinct electrical advantages. First, since the ring sends messages from one point to only one point, the termination impedance of the transmission medium can be tightly matched, eliminating most electrical reflections.

Second, the ring may easily make use of the ultra-high bandwidths afforded by optical fiber technology. It should be noted that using fiber optics in a bus architecture may be impractical because of the possible lack of suitable low-cost taps.

Fiber Optics, too

Nevertheless, a very high-speed ring specification is currently being developed by ANSI. The ring is called the Fiber Distributed Data Interface (FDDI). The FDDI can be configured as dual 100 Mbit per second rings, supporting two classes of stations. See Figure 5 (a) for details.

Class 'A' stations have access to both rings, and therefore have 200 Mbit per second bandwidth at their disposal. Class 'B' stations may only access a single ring and hence their maximum bandwidths are restricted to 100 Mbits per second.

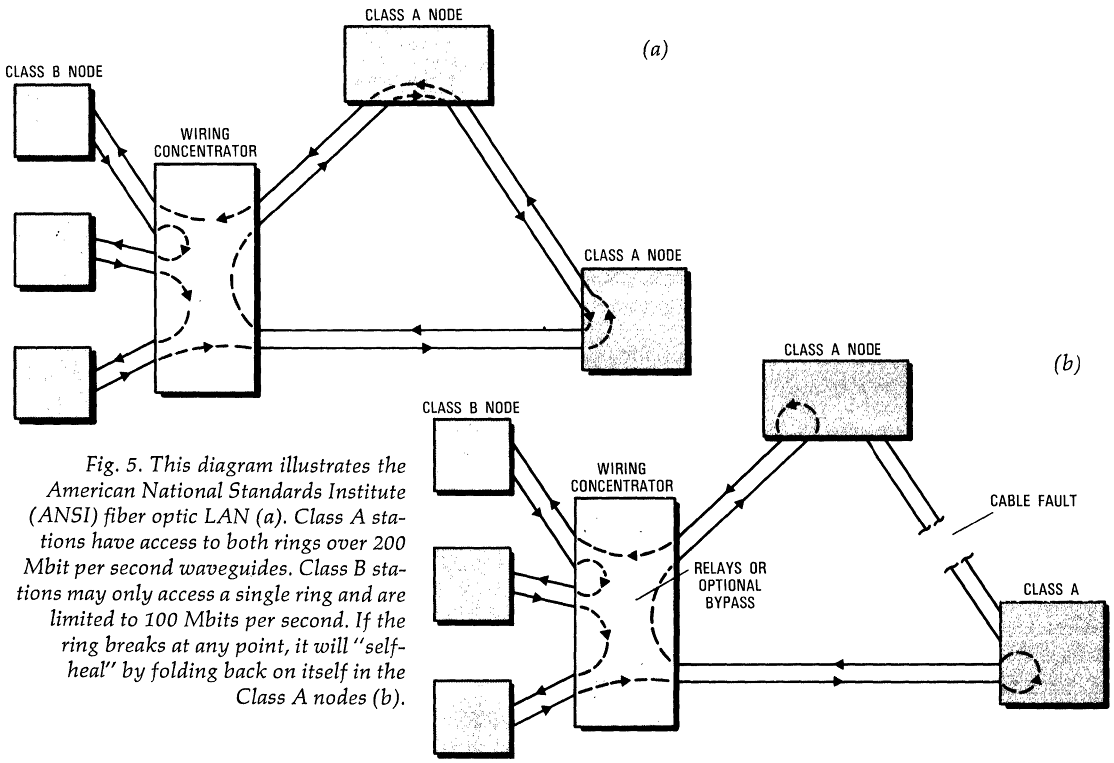


Fig. 5. This diagram illustrates the American National Standards Institute (ANSI) fiber optic LAN (a). Class A stations have access to both rings over 200 Mbit per second waveguides. Class B stations may only access a single ring and are limited to 100 Mbits per second. If the ring breaks at any point, it will "self-heal" by folding back on itself in the Class A nodes (b).

The Fiber Distributed Data Interface uses a timed token scheme for node access. The timed token is held by a node for a programmable amount of time. With the FDDI system, the token holding time is divided to provide synchronous bandwidth and optional asynchronous bandwidth.

The token time for synchronous operation is guaranteed for each node, thereby assuring enough bandwidth for real-time critical applications such as file servers, data base machines, real-time voice, and serial control buses. If a node also has applications that do not require a minimum specified bandwidth, then the asynchronous option may be used in addition.

Asynchronous bandwidth can be subdivided into a series of priority levels. Devices using asynchronous bandwidth are allowed access to the media based upon the amount of time that a node has remaining before it must relinquish the token.

In other words, if a node has transmitted all its high-priority guaranteed-bandwidth messages, and still has time left over, it may transmit lower-priority messages. The order of transmission of these messages depends on the priority of the message, and the time remaining before the token must be sent to the next node in the net.

The FDDI ring is fully distributed such that no individual node has responsibility for ring initialization and error recovery. Any node of the network may arbitrate for and assume that responsibility.

The architecture of FDDI also allows for a self-repair feature that is implementable with dual rings. If the ring breaks at any one point, it will "self heal" by folding back on itself in the class A nodes immediately preceding and following the break. The overall result is enhanced reliability. See Figure 5 (b).

More Tradeoffs

Rings have some disadvantages in practical implementations. Every node on a ring must have bypass provisions in case of node failure. For cables this would probably be a relay, and for optical waveguide media the mechanism would be an optical bypass of some sort.

To solve the problem of power distribution to these bypasses and to provide a better maintenance element, the bypasses are usually grouped in a wiring concentration. What results is a ring that outwardly resembles a star architecture, as shown in the class 'B' nodes of FDDI. ■

Use a CRT-controller chip to mix text and graphics

You needn't sacrifice a CRT-controller (CRTC) chip's performance when adding bit-mapped-graphics capability to a computer display system, as a CRTC-based graphic-display-block implementation scheme demonstrates.

Mark S Young, *Advanced Micro Devices Inc*

You can use a VLSI text controller to display text and graphics simultaneously without compromising system price or performance. The Am8052 CRT controller (CRTC) overcomes the limitations of two schemes commonly used to mix text and graphics. On the one hand, fixed or programmable character sets prove clumsy for graphics implementation; on the other hand, bit-mapped pixel-plane solutions, the best for high-resolution graphics, require excessive pixel manipulations when handling text.

Chip allows graphics windows within text

Mixing graphics and text using a CRTC such as the Am8052 involves creating graphic display blocks (GDBs), or graphics windows, within the text. A GDB is a bit-mapped graphics window—that is, a window whose pixel-by-pixel representation is stored in RAM—composed of contiguous character locations that form a rectangle on the display screen (**Fig 1**). Setting up one or more bit-mapped windows in a GDB display memory and then manipulating the pointers from the Am8052's text-display list (see **box,** "Details of the CRT-controller chip") makes the bit-mapped window appear on the screen. With these graphics windows integrated into the normal CRTC display lists, the CRTC can handle all display functions without extra hardware.

Though functionally similar to bit-mapped systems, an Am8052-based GDB system offers the best features of the bit-mapped graphics and text worlds. Bit-

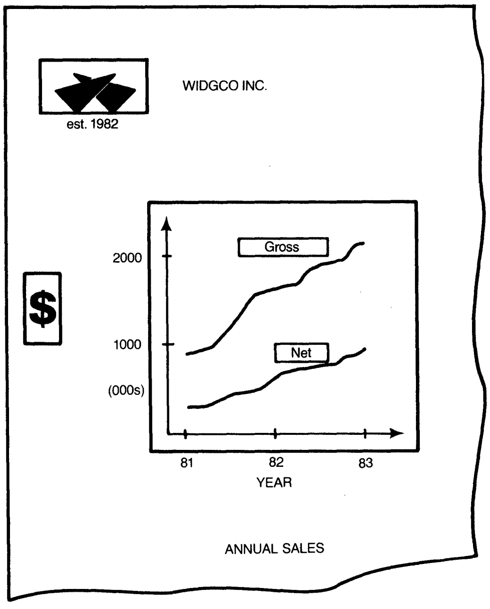


Fig 1—You can mix graphics and text *by designing an Am8052 CRT-controller-based graphic-display-block (GDB) system.*

Details of the CRT-controller chip

A general-purpose CRT controller (CRTC) for raster-scan displays, the Am8052 CRTC chip, supports high-resolution monitors at pixel rates to 100 MHz when coupled with the Am8152/3 video system controller (VSC).

The Am8052's on-chip interface circuitry supports the 8086, Z8000 and 68000 μP families. Its advanced text features, such as proportional character spacing and automatic concatenation of trailing blank pixels to each character, support the needs of modern text processors. Moreover, the chip allows the overlay of text windows onto background text.

The Am8052 also supports screen formats to 132 characters per row, and three internal 132-character row buffers allow flicker-free split-screen smooth (or soft) scrolling at user-programmable rates without external logic or software overhead. (Split-screen scrolling refers to scrolling background text while freezing window text, or vice versa.) This soft scrolling, which involves moving data up or down in 1-pixel increments, eliminates the jerky display that results from hard scrolling, which moves characters in 1-row increments.

Fig A highlights the function of the chip. It's called a text controller because it handles more than simple video-timing and screen-display functions. For example, it includes a text-data and control-information structure that eases text manipulation. An on-board linked-list-management unit interprets the text-display list and controls the function of the on-board DMA unit.

Divided into background- and window-data structures **(Fig B),** the display list includes the main definition block (MDB), the character (or row) code, the character-attribute/display-option information, the row-control block (RCB), the window-definition block (WDB) and the window-control block (WCB).

The MDB contains screen-control data, including the smooth-scroll rate, scroll direction (up and down), cursor position, the cursor- and character-blink rate, and links (pointer addresses) to the row information. The row code contains eight bits of character-code information for the font ROM. The row attributes contain four bits of user-definable data as well as text function bits that control such optional features as character highlighting, reverse video, superscripting, subscripting and character underlining. Multiple attribute functions can be used simultaneously.

The Am8052 automatically executes the link-list data structure that the user sets up. You can manipulate the contents of the row character-code and attribute structure as data strings, each of which contains one to 132 display characters. Connecting the strings allows the display of a row of characters, whose order the string arrangement dictates. Operations such as inserting new data rows and deleting or inserting characters, words or strings therefore only require string-pointer manip-

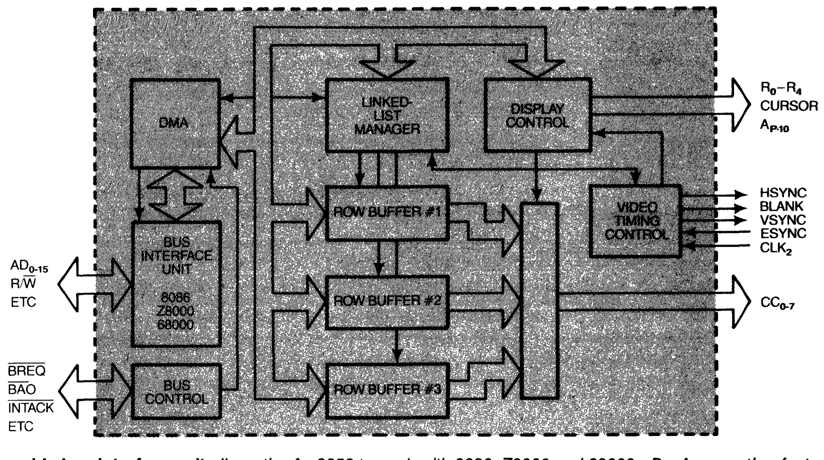


Fig A—An on-chip bus-interface unit *allows the Am8052 to work with 8086, Z8000 and 68000 μPs. Among other features, a DMA unit handles 5M-byte/sec data transfers, three 132-character row buffers support smooth scrolling, and a link-list manager handles character and attribute manipulations.*

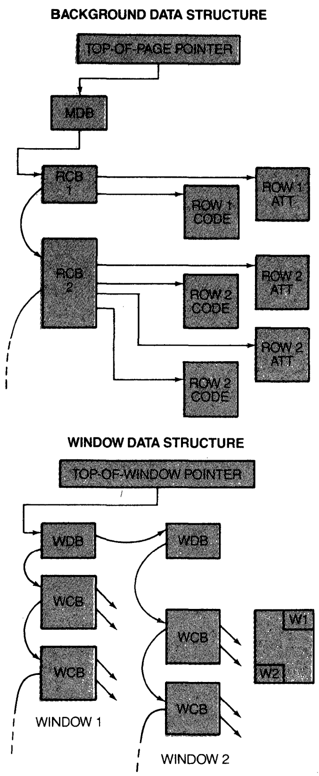


Fig B—The text-display list *supported by the Am8052 consists of background- and window-data structures, which in turn include the main definition block (MDB), the character code (labeled ROW CODE here), the character-attribute/display-option information (labeled ROW ATT here), the row-control block (RCB), the window-definition block (WDB), and the window-control block (WCB).*

ulation, and not complete data rows. **Fig C** illustrates the linked-list control-structure alteration required to edit a display row.

The attribute-code strings, which complement the character-code strings, allow each character to have a unique attribute word, although a block of characters may share a single attribute, which reduces the Am8052's bus-transfer time. Setting a specified option bit in the attribute word enables attribute functions. For example, enabling the subscript-option bit causes the Am8052 to alter the corresponding character's ROM pixel row and addressing to shift the character down on the screen.

The RCB, fetched at the beginning of each row, contains as many as 10 bits of optional user-defined attributes. These appear as output on the Am8052's attribute pins during horizontal retrace. The RCB also contains information like character height for the row and super- and subscript positioning data. You can alter such attributes row by row.

The WDB enables another optional feature—the creation of special independent screens (called foreground) overlaid on various sections of the main display (the background). Linked to the WDB, the WCB performs the operations of the RCB within the overlaid miniscreen. This windowing function allows you to create such a miniscreen without disrupting the data that normally appears in the window space. When the window is removed, the normal screen reappears without the need to recreate the space that the window formerly occupied.

In addition to its link-list manager, the Am8052 includes a DMA unit with a 16M-byte addressing capability and a transfer rate greater than 5M bytes/sec. The Am8052's DMA unit permits bus pre-emption by other devices.

The Am8052's display and video-timing controllers provide all the necessary video timing and character-code generation for the character-font ROM. They furnish scan-row, character-code and attribute information for each character and generate the correct

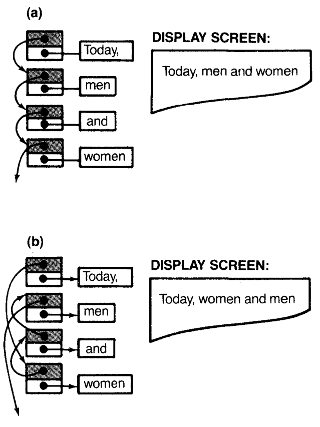


Fig C—In systems based on the Am8052, *simple pointer manipulations can alter displayed data. Here,* ***(a)*** *shows a row of text and the corresponding link-list control structure;* ***(b)*** *shows the edited text that results from the indicated pointer changes.*

Blank, HSYNC and VSYNC timing. On-board parameter registers allow programming of many video-controller functions to suit particular applications. For example, normally only four of 12 attribute bits for each character code are user definable. However, using the attribute-redefinition register, you can redefine eight predefined attributes.

A common problem with CRTCs lies in the maximum width and height of the character-display matrix. The Am8052/8152 chip set limits the character matrix to 17 pixels in width and 32 pixels in height, although some applications would benefit from a larger matrix. The Japanese Kanji character set, for example, requires at least a 24×24 matrix to display complicated characters correctly. with a minimum amount of additional software, the Am8052 can handle such tasks, permitting matrices as large as 34×32.

mapped systems, for example, offer such features as smooth scrolling and text sub- and superscripts—at great cost in terms of graphics-processing power. However, because GDBs use the same control structure as text-processing tasks, implementing smooth scrolling and other functions requires no extra effort.

GDBs also ease graphics-memory management. Because a pointer system accesses GDBs, you can put the graphics window data virtually anywhere within display memory. Moving the window only requires manipulation of a few pointers; thus, graphics blocks can be rapidly moved from one screen position to another. With a bit-mapped system, moving a window requires moving an entire window's pixel representation from one place in display memory to another. In addition, a GDB system allows a single graphics image to appear simultaneously on several parts of the display, because several GDBs can point to the same graphic image's representation in memory.

GDB system effectively uses memory

If you don't require full bit-mapped capability, the GDB-based system allows you to reduce display memory size. In contrast, full bit-mapped systems require maintenance of a contiguous display memory, and such systems cannot make use of RAM wasted in the pixel plane. In the GDB system, the extra RAM can store preset patterns, unique character sets, or frequently used graphics images such as corporate logos.

The effect of using this extra RAM is similar to that provided by dual graphics display planes in bit-mapped systems. Such dual planes allow alteration of one plane while the other is displayed. The GDB system approximates the dual-plane system's performance without the latter's cost. And because, as in the dual-plane system, a GDB system's graphics need not be displayed as they are drawn in memory, the GDB system processor can be relatively slow in comparison with its apparent throughput.

Implementing the GDB interface

Now consider the GDB system's implementation. As the **Fig 2** block diagram shows, the Am8052-based GDB system closely resembles a bit-mapped system's architecture. However, rather than include the Am8052 and a character-font ROM, a bit-mapped system has a video-timing controller/address generator, and all video data comes from bit-mapped memory. In addition, a bit-mapped system requires implementation of the entire memory plane. An Am8052-based GDB system

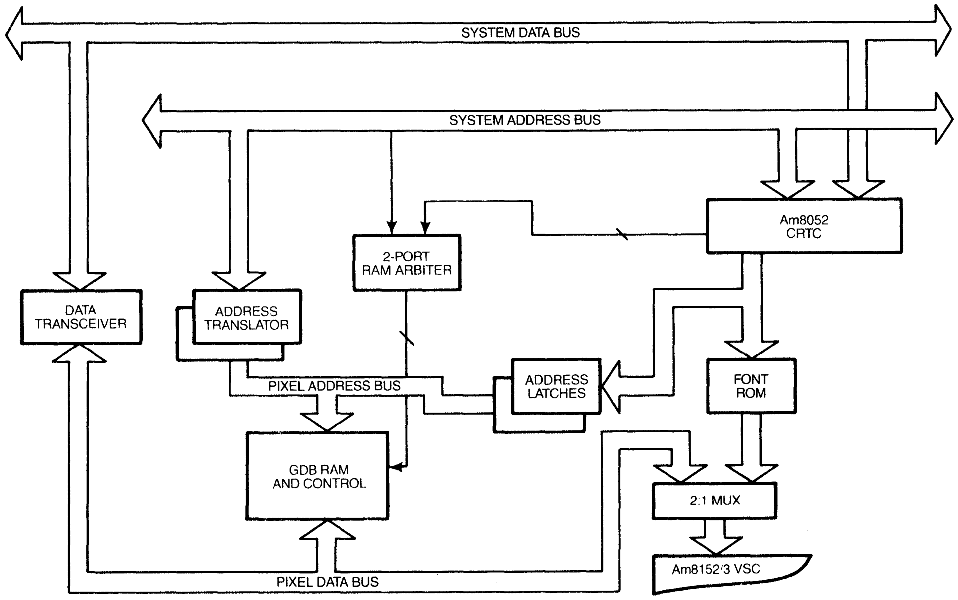


Fig 2—The system architecture *for a GDB closely resembles that of a bit-mapped system. Unlike a bit-mapped system, though, this GDB system includes a character-font ROM.*

needs only enough RAM to hold the required graphics windows. The Am8152/53 chip functions as the video shift register and pixel timing controller required in a bit-mapped implementation.

Integrating graphics and text display in an Am8052-based system involves dealing with three problems:

- Generating the necessary graphics RAM addressing (in a format compatible with an X-Y-type pixel plane)
- Integrating the graphics windows into the normal text display list
- Telling the Am8052 when to display graphics and when to display text.

To understand the generation of an X-Y addressing scheme for a graphics window, first look more closely at the graphics pixel plane's specification. The characters per row, number of rows and character-display matrix size determine the horizontal and vertical pixel resolution of a text display. When implementing mixed-mode text and graphics, the graphics pixel must be the same size as the character pixel, because GDBs directly replace the area normally allocated to text display. (In some graphics/text machines, such as the IBM Personal Computer, character- and graphics-pixel sizes differ. For text only, the PC supports a 720×350-pixel screen. However, in the high-resolution graphics mode—in which it displays text as graphics patterns—the PC's resolution is only 640×200 pixels.)

Calculate obtainable resolution

Depending on the mixed-mode text/graphics requirement for same-size graphics and text pixels, you can readily calculate the obtainable graphics resolution based on the character-display attributes. For example, a typical personal computer supports a character display of 80 columns×25 lines with a character-display matrix measuring nine pixels wide by 14 pixels high. In such a system, the character pattern matrix typically measures 7×9 pixels; the extra pixels in the 9×14 character-display matrix space the characters (horizontally and vertically) and permit such features as underlining, subscripting and superscripting. Converted into pixel-resolution terms, such a computer (including the IBM Personal Computer) has a theoretical graphics resolution of 720 pixels horizontally (80 characters times 9 pixels per character) and 350 pixels vertically (25 rows × 14 pixels per row).

The standard method for addressing a bit-mapped plane dictates that the memory plane be closely oriented toward the X-Y coordinate system used by the graphics algorithms. Because RAM chips use addressing schemes based on the binary number system, the dimensions of the graphics display area must be fitted to the nearest power of 2 equal to or greater than the desired display dimensions. In the example discussed in the last paragraph, the nearest power of 2 equal to or greater than the 720-pixel horizontal display size is 2^{10}, or 1024. Thus, 10 address lines perform the column addressing for each pixel in a row.

The memory plane must also contain a sufficient number of pixel rows. Because GDBs are aligned on character rows, the character-display height (14 pixels) must also fit within a power of 2—2^4, or 16, in this case. This requires a minimum of 2^4×25, or 400, pixel rows. Supporting a complete plane therefore requires 512 rows (the nearest power of 2 greater than 400), or another nine address lines. Thus, although an 80-character×25-line display with a 9×14 character-display matrix would require 80×9×25×14 (252,000) pixels, the power-of-2 requirements that facilitate memory interfacing increase the number to 1024×512, or 524,288, pixels.

Fig 3a shows an 80×25-character display bit-mapped within a 1024×512-pixel X-Y coordinate system. Because pointers specify the GDBs, the extra RAM can store frequently used patterns or be used to draw figures before they're displayed.

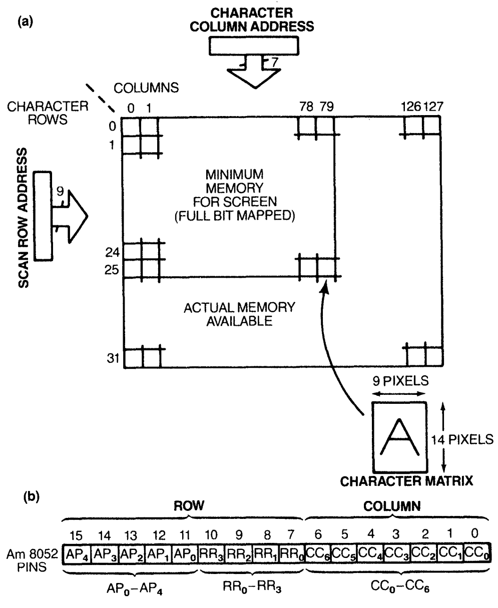


Fig 3—In a graphic-display-block system, *RAM addressing considerations mandate that an 80×25-character display be bit-mapped within a 1024×512-pixel plane* ***(a)****; extra RAM can store frequently used patterns. RAM address lines correspond to Am8052 pin designations* ***(b)****.*

The GDB approach uses memory effectively

Before looking at the memory and hardware architecture of the Am8052-based GDB bit-mapped display controller, consider the limitations imposed by video-timing requirements. A 720×350-pixel display requires 864 horizontal pixels to allow a 20% margin for vertical retrace and horizontal blank time. This yields a virtual display size of 302,400 pixels. Refreshing the screen at 60 Hz (noninterlaced) to obtain flicker-free viewing requires display of 18 million pixels per second, which translates to a 55-nsec/pixel access time.

Shift register speeds pixel access

To overcome the system memory's performance limitations relative to this pixel-update requirement, video systems fetch video data in parallel and load it into a high-speed serial shift register. For the example presented here, the memory plane is arranged as 9-bit-wide data, with nine 16k×1-bit RAMs used to implement the 9-bit word. This size simplifies interfacing to the Am8152 video system controller, and because the character matrix width is nine bits, the 9-bit size simplifies the graphics CPU's tasks. As an additional benefit, the 9-bit-wide arrangement reduces the column-address-line requirements from 10 lines to seven, because it allows addressing of each group of nine pixels, as opposed to requiring individual addressing of each pixel.

This design thus requires a total of 16 address lines (seven for column addressing and nine for row addressing). **Fig 3b** shows these lines and their corresponding

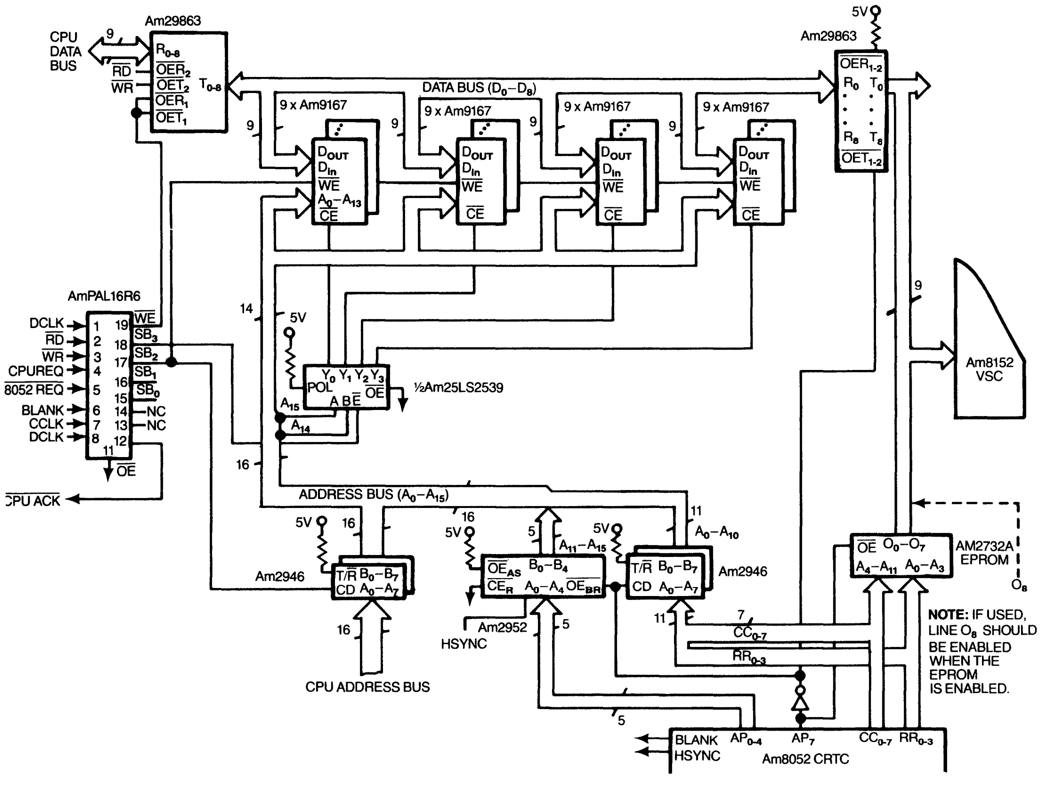
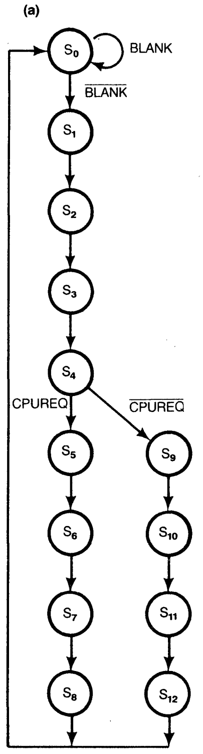


Fig 4—This GDB system uses four banks *of nine 16k×1-bit static RAMs to implement a full bit-mapped plane. An Am16R6 PAL handles the 2-port access-controller and timing-generator function, giving the CPU access to display RAM during 50% of each character-clock cycle.*

(a)



SB_3	SB_2	SB_1	SB_0	ASSIGNED STATE
0	0	0	0	S_5
0	0	0	1	S_6
0	0	1	0	S_7
0	0	1	1	S_8
0	1	0	0	S_1
0	1	0	1	S_2
0	1	1	0	S_3
0	1	1	1	S_4
1	0	0	0	S_0
1	0	0	1	—
1	0	1	0	—
1	0	1	1	—
1	1	0	0	S_9
1	1	0	1	S_{10}
1	1	1	0	S_{11}
1	1	1	1	S_{12}

(b)

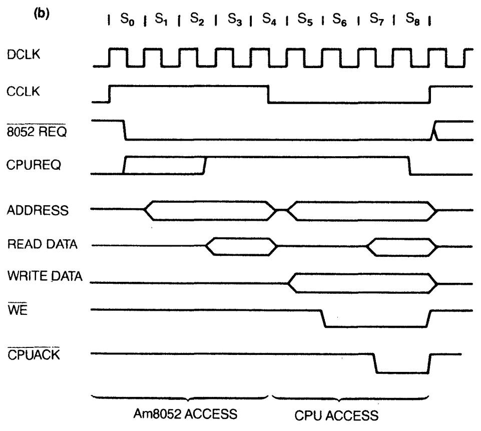


Fig 5—The state (a) and timing (b) diagrams *describing the operation of* ***Fig 4's*** *circuit allow direct derivation of the PAL equations. CCLK is the 2.016-MHz character clock; DCLK is the 18-MHz dot clock.*

Am8052 pins. This design uses the normal 8-bit character code to provide column-address values of 0 to 79 (from pins CC_0 through CC_7), and you must replace the ASCII character codes with the column value whenever a GDB replaces a character's column position within a row. For instance, if the right half (columns 40 through 79 in this example) of each of the top three screen rows becomes part of a GDB, then the character codes in columns 40 through 79 on each of the three top rows has to change to represent the column number of the character location being replaced.

The nine bits for row addressing come from two places: five bits from the row redefinition attributes and four bits from row-display-counter pins RR_0 through RR_3.

One user-definable attribute in the text-attribute data indicates whether a character or part of a GDB is being displayed; that is, it indicates whether to interpret the character code as an ASCII code or as a column-address value for a GDB. This attribute, which appears as an output on the Am8052's AP_7 line at the same time as the character code, controls multiplexing between the character-font ROM and the GDB display RAM. You must ensure that this bit is set whenever a character position is designated as part of a GDB. This attribute bit, which must be cleared when a normal character is reinstated in place of the GDB segment, resets when a legitimate character code replaces the old GDB column value.

Hardware implementation

The GDB graphic memory plane can consist of static or dynamic RAMs. Using static RAMs greatly simplifies the control timing and allows you to use slower, cheaper parts while maintaining a graphics interface that allows CPU access 50% of the time. This static-RAM approach employs four banks of nine 16k×1-bit RAMs to implement a full bit-mapped plane. If you need only a programmable character set or limited bit-mapping capability, this scheme allows you to put in only the RAM you need for your application. Each 9-chip bank can create a display window of eight lines of 80 characters each or 16 lines of 64 characters each, or store a 1024-character programmable character set.

Fig 4 shows the hardware required to implement the static-RAM scheme. In addition to the RAM (with 120-nsec access times) and Am8052, the circuit includes an Am16R6 PAL (which handles the 2-port access-controller and timing-generator function) as well as bus transceivers, address latches, and half of a decoder.

The 2-port RAM-access controller arbitrates the Am8052's and the CPU's access to display RAM to prevent screen glitches. It gives the CPU access 50% of the time, as **Fig 5** indicates. The character clock

(CCLK) runs at 2.016 MHz, leaving just under 500 nsec per cycle for the Am8052 (which requires only half this time) to access RAM. Using RAM with 200-nsec or shorter access times leaves an adequate margin for transceiver turn-on and propagation delays.

Fig 5b illustrates the 2-port controller arbitration. Am8052 requests are synchronized to the 18-MHz dot clock (DCLK) and have a worst-case delay of 35 nsec from the rising edge of the CCLK. Only the CPU requests need to be synchronized to ensure reliable arbitration. All Am8052 accesses occur during the high portion of the CCLK; pixel data is valid on the falling edge of the CCLK. The CPU is allowed access only during the low portion of the character clock. Because of the synchronization delay, CPU requests must arrive at least two DCLK edges before the falling edge of the CCLK; otherwise, the CPU must wait until the next CCLK cycle to access the GDB RAM. This timing is enforced even when the Am8052 requires no accesses, keeping the arbitration scheme simple. You can derive the PAL equations directly from **Fig 5**'s state and timing diagrams.

The dynamic RAM approach

An alternative to the static-RAM approach is to use denser dynamic RAMs. To handle those devices' refresh requirements, you could use an Am8150 display-refresh controller, which provides all the refresh control and logic timing necessary to use dynamic RAMs in a GDB memory plane. Using the Am8150,

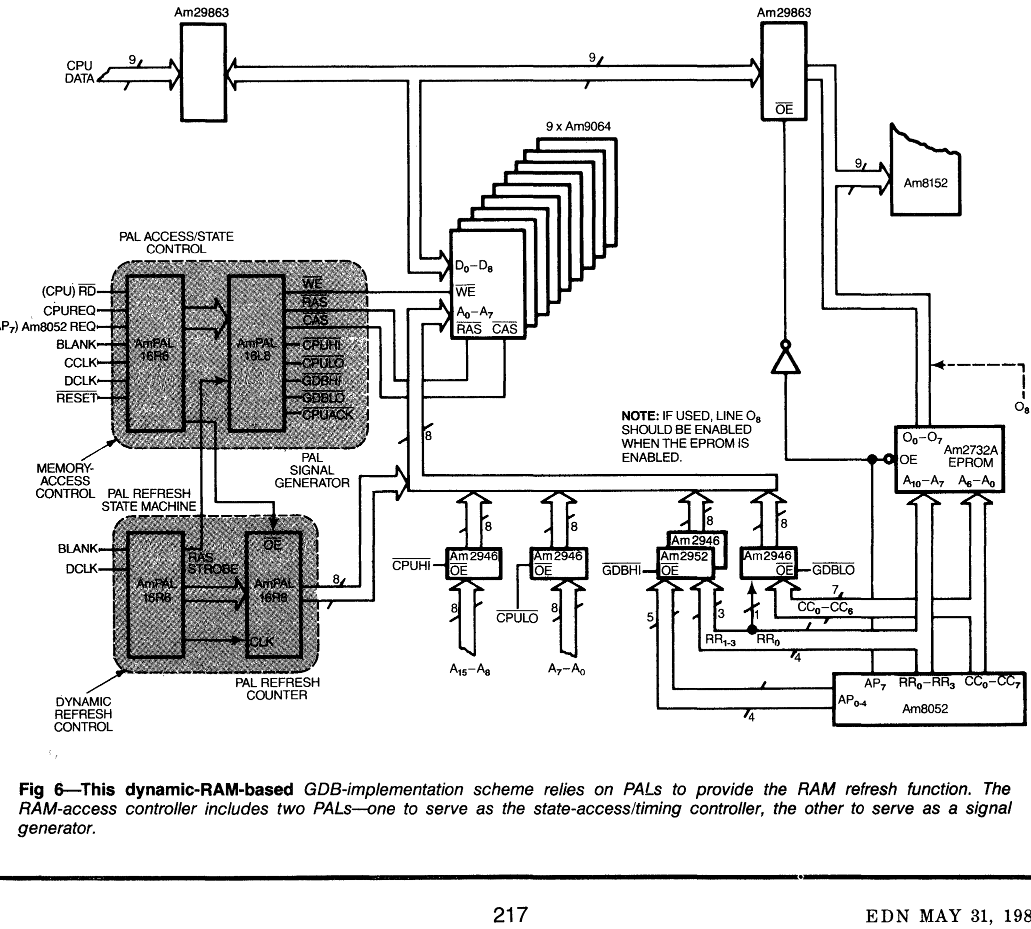


Fig 6—This dynamic-RAM-based *GDB-implementation scheme relies on PALs to provide the RAM refresh function. The RAM-access controller includes two PALs—one to serve as the state-access/timing controller, the other to serve as a signal generator.*

Static or dynamic RAM can serve as the graphic-memory plane

however, might require modification of the Am8052's attribute function to indicate the beginning and end of a graphics window on a particular row of text. (Recall that, normally, the attribute function defines which character is part of a GDB.) Alternatively, you could double the character-clock rate and feed it into the Am8150's character-clock input to allow interleaved video and CPU access to the GDB RAM.

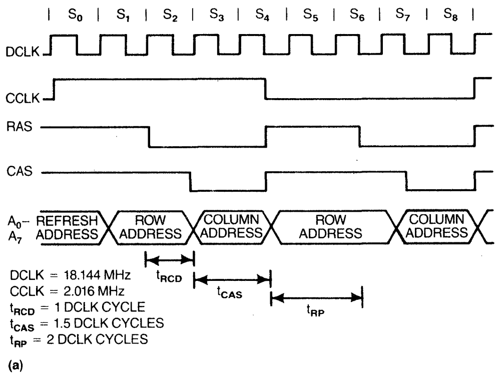
Fig 6 illustrates another dynamic-RAM implementation method—one employing a PAL-based refresh scheme. Here, a minor modification divides the Am8052's 16-bit address into two groups of eight address lines. The access controller includes two PALs; one serves as the state-access/timing controller, and the other is the signal generator. As in the static-RAM example, the timing setup allows interleaved Am8052 and CPU access; the circuit requires dynamic RAMs with 120-nsec or less access times.

Fig 7a illustrates the timing of **Fig 6**'s circuit. Although the PAL equations are more complicated than those in the static-RAM example, you can derive them directly from the timing diagram. Close scrutiny of this diagram shows that the refresh address is always on the address bus whenever Am8052 or CPU addresses are not, a condition that prevents address lines from floating and possibly subjecting the dynamic RAMs to glitches when they're not being accessed.

PALs handle refresh

Two other PALs handle refresh. One of these PALs acts as a refresh timing controller, while the other serves as an 8-bit refresh address counter. The Am9064A 64k×1-bit dynamic RAMs require 128 refresh cycles every 2 msec. The only time that Am8052 accesses cannot occur is during the horizontal retrace and blanking period, so all refreshing can be performed when the Blank line is active. A refresh operation occurs after display of every 42 rows (or 42 occurrences of the Blank signal), which requires that the refresh controller perform four refresh cycles per HSYNC period to keep the memory valid. To simplify timing further, the CPU is denied RAM access during the horizontal retrace and blanking period. You can derive the refresh-timing diagram and PAL equations directly from the dynamic-RAM data sheet.

Other dynamic-RAM approaches allow you to use slower devices. One such approach involves limiting CPU access to the horizontal retrace and blanking time, and to those times when the Am8052 doesn't require access to the GDB RAM. **Fig 7b** illustrates the timing for such a technique, which allows use of dynamic



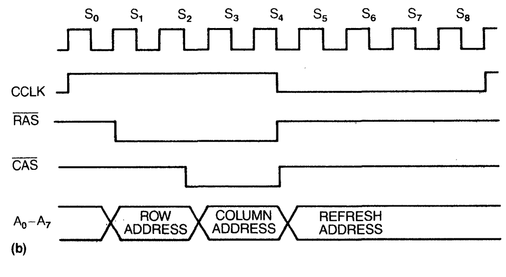


Fig 7—The logic equations for the PALs *that compose* ***Fig 6****'s access controller can be derived from the timing diagram in* ***(a)****. Part* ***(b)*** *shows the timing for a technique that allows use of slower dynamic-RAM chips.*

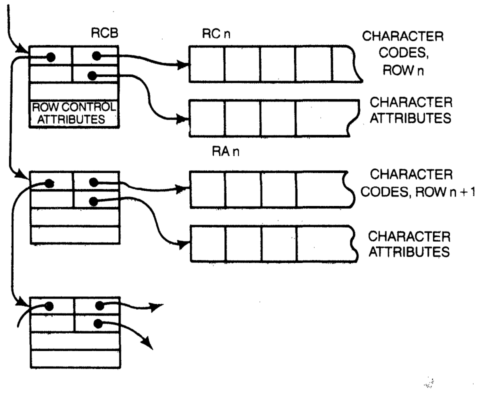


Fig 8—The character/text-handling software *for an Am8052-based GDB system involves the manipulation of linked lists, whose structure is shown here for normal text display.*

Dynamic-RAM implementation employs PAL-based refresh scheme

RAMs with access times as long as 450 nsec. Note that with this approach, CPU access can be severely limited, depending on how much of the screen has been turned into a bit map.

Another method involves creating two banks of dynamic RAMs and making the Am8052 fetch two characters at a time. This method allows the CPU to access the GDB RAM every other CCLK cycle, but it imposes some addressing limitations on the GDBs and involves a more complicated timing scheme.

Software considerations

Typical of most modern designs, the support software for implementing graphics with GDBs is a large portion of the project. The required software falls into three distinct sections:

- Character/text-handling routines
- GDB-creation and -handling routines
- Graphics routines for drawing images in the GDB areas allocated in the display memory.

The character/text-handling software applies as well to Am8052-based systems that don't use GDBs, and it involves the manipulation of linked lists. **Fig 8** shows the structure of these linked lists for normal text display. The software must be able to modify any character displayed on the screen by changing character codes or manipulating attributes.

The GDB-creation and -handling software is a variation of the text-handling software. In addition to performing the text-handling tasks, it must determine the GDB window size, set up the correct address pointers to the GDB RAM and update the linked list with the correct attribute bits enabled or disabled.

Draw a corporate logo

For example, **Fig 1**'s corporate logo, as detailed in **Fig 9a,** is six characters wide and three characters high, and the upper-lefthand corner corresponds to a character X, Y position of 2, 4, using the coordinate convention of **Fig 3a.** Placing the equivalent bit-mapped GDB in the graphics memory involves minor modification of the row-redefinition blocks, the character codes and the character-attribute bits of the affected screen display rows.

In this corporate-logo example, the row-redefinition blocks for text rows 4 through 6 require that the binary values 4, 5 and 6 be placed in the attribute bits AP_0 through AP_4 of the corresponding rows. Then, within each partial GDB row (rows 4 through 6 in this example) the character codes for each column affected (columns 2 through 7) must be changed to the equivalent column value (2 through 7, respectively). Finally, the attribute bit AP_7 must be set to denote each character cell as part of the GDB.

Figs 9b and **(c)** show the resulting transition from a normal text display to a display that includes **(a)**'s GDB. Once you decide on a GDB's size, manipulating the display linked list involves only minor effort. Disabling a GDB requires that you clear the GDB Enable bit in each character in the (soon-to-be-destroyed) GDB and replace the character column values with their normal character values. Because even a text-only Am8052-based system requires all the display list-manipulation software, the GDB-handling capability requires no significant overhead.

GDB systems can use standard software

The graphics software supporting an Am8052-based GDB system is no different that that used with other hardware configurations. You can use standard industry algorithms in such a system, and commercial packages such as VISI-ON or Microsoft WINDOWS can be accommodated with only simple rewriting of the low-level driver software that talks directly to the hardware. Because all graphics packages require unique software drivers for physical pixel manipulation, such effort does not represent a great deal of overhead.

The most complicated software function involved in GDB use is translating normal X-Y pixel positions into the correct RAM addresses for the graphics processor. The algorithm for this task depends on the character-font size chosen and the physical setup of the GDB RAM. Addressing a particular pixel involves first calculating the X-Y position relative to the GDB. For example, if a GDB is 10 characters wide and five characters high (with a 9×14 character matrix), the X-Y positions are calculated relative to a bit-map window measuring 90×70 bits. After calculating these coordinates, you may have to add an offset to compensate for the GDB's actual position in display RAM. Remember that, unlike standard bit-mapped systems, GDBs are not constrained to a particular physical location in display memory.

The general formulas for calculating a GDB pixel's X-Y coordinates, based on X_{COORD} and Y_{COORD} (the pixel's X-Y coordinate values in the idealized bit-mapped plane), are

$$X_{FINAL} = X_{COORD} \text{ DIV CW}$$

$$X_{BIT} = X_{COORD} \text{ MOD CW}$$

$$Y_{FINAL} = (Y_{COORD} \text{ DIV CH}) * N + (Y_{COORD} \text{ MOD CH}),$$

where CH and CW are character height and width (in pixels), respectively, N is the power-of-2 alignment factor, DIV represents integer division with no remainder, and MOD calculates only the remainder of integer division.

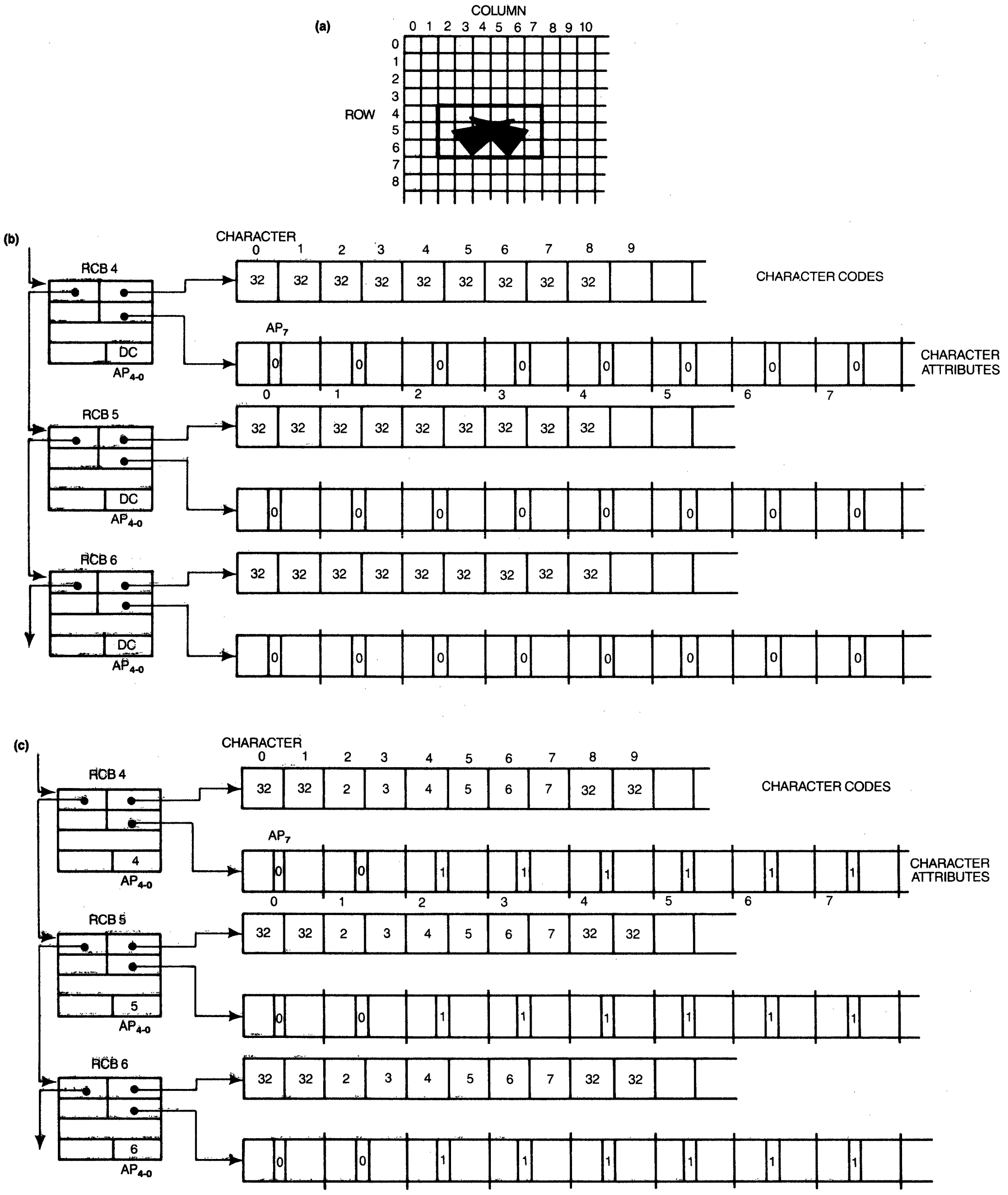


Fig 9—Displaying a corporate logo (a) *as a GDB involves modifying the character codes and AP$_7$ bits from their normal text formats* **(b)** *to their graphics format* **(c)**. *Note that during GDB display, the character codes within the graphics window take on their column values (4 through 7 here) in place of their ASCII codes, and the corresponding AP$_7$ bits are set to One. (Note that 32 is the ASCII code for a space.)*

Using slow dynamic RAMs can severely limit CPU access

Because the GDB plane is not generally arranged in a bit-addressable manner, the X_{BIT} value in this formula determines which pixel in the word fetched is the pixel to be manipulated. Once X_{FINAL} and Y_{FINAL} are known, the offset address to the GDB window's 0, 0 location is added to the 16-bit address, whose upper nine bits are the Y_{FINAL} value and whose lower seven bits are the X_{FINAL} value. Modifying a single pixel within the pixel word fetched requires a graphics-processor read-modify-write cycle.

A look at these formulas for calculating GDB-memory coordinates illustrates why display manufacturers often choose particular character-font sizes and bit-map resolutions. For example, the IBM PC's 8×8 graphics-character matrix allowed the designers of that system to replace the DIV, MOD and multiplication instructions with logical-shift and AND functions. Although modern 16-bit µPs, such as the 8086 and Z8000, have multiplication and division built into their instruction sets, they take considerably longer to calculate the translated X-Y coordinate values using those arithmetic instructions than they do using simple shifts and logical operations. Moreover, given the continued widespread use of 8-bit processors, many designers prefer to avoid multiplication and division requirements whenever possible.

Look-up table avoids translation

A look-up table that a CPU can use to find translated coordinates is a simple alternative to using the coordinate translation equations just discussed. Such a technique requires approximately 1k words of memory: 350 words to generate the nine bits of the Y_{FINAL} portion of the GDB RAM address (one word for the beginning address of each pixel row) and 720 words to generate all 720 permutations of the horizontal-offset X_{FINAL} and X_{BIT} coordinates.

Figs 10a and **(b)** show the look-up table's configuration. Part **(c)** shows the X_{FINAL} and Y_{FINAL} values assembled into a 16-bit address—which refers only to the GDB memory space of 64k-pixel words. You must add an offset to this address to make it access GDB RAM in the system address space.

The slowness of pixel modification is one problem of a display RAM such as the one described in this article. Because high-speed-RAM technology has not demon-

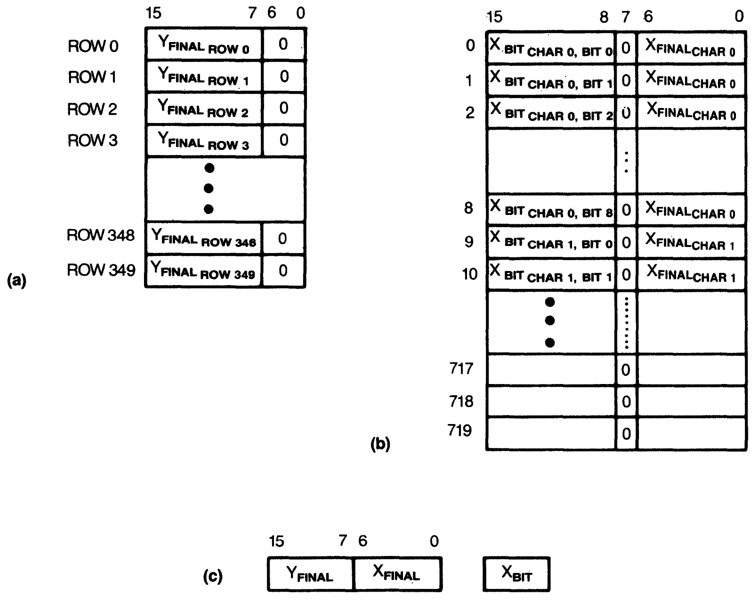
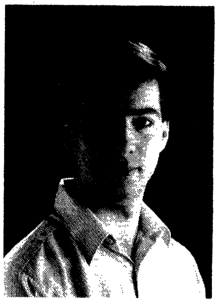


Fig 10—A look-up table *requiring approximately 1k words of memory can replace coordinate-translation equations in GDB systems. The table illustrated here is divided into Y-coordinate* ***(a)*** *and X-coordinate* ***(b)*** *subtables. The resulting values combine to form a 16-bit address and an* X_{BIT} *value* ***(c)****.*

strated a cost-effective way to address 700,000 to 1,000,000 pixels individually, slower RAMs designed for parallel operation are used, and processor bit-test and -set instructions are used to perform the read-modify-write cycles required to modify a single pixel. This approach yields a low-cost alternative to specialized pixel-addressing and -manipulation hardware. **EDN**

Author's biography



Mark S Young, a product planning and applications engineer at Advanced Micro Devices Inc (Sunnyvale, CA), specifies and designs µP peripheral chips. He previously worked for Arthur D Little and Commodore Business Machines. A member of the IEEE, Mark received a BA degree in computer science in 1981 from the University of California, Berkeley.

L'architettura dell'Am29116 utilizzata in un processore grafico bit-mapped, permette di tracciare vettori, caratteri e poligoni in maniera interattiva e con alta dinamica di movimento.

W. Miller, R. Mazzoni

NUOVE ARCHITETTURE PER ACCELERARE LA GESTIONE DEI PIXEL NELLE APPLICAZIONI GRAFICHE

Ci sono tradizionalmente tre tipi di tecnologie di visualizzazione grafica su tubo a raggi catodici (CRT): il sistema stroke, il sistema raster scan conversion ed il sistema raster scan bit-mapped (Tabella 1). In un sistema stroke, i vettori sono rappresentati tramite le coordinate di un punto ed istruzioni di spostamento da un punto ad un altro. Queste istruzioni vengono usate per deviare il pennello elettronico sul CRT. Una linea può essere spostata o ruotata molto rapidamente semplicemente ricalcolando le coordinate del punto terminale. Si può ottenere una risoluzione molto elevata. Sistemi a vettori con una risoluzione spaziale di 4096 x 4096 pixel non sono una rarità in questo ambito. Tuttavia i complicati circuiti di deflessione richiesti li rendono costosi. Poichè l'immagine del vettore deve essere ridisegnata continuamente (almeno 30 volte al secondo), c'è un limite alla massima complessità raggiungibile nell'immagine, fissato dalla massima velocità di tracciatura che il sistema stroke può tollerare senza produrre sfarfallìo. Un tubo di tipo Direct View Storage Tube (DVST) può trattenere l'immagine senza bisogno di ridisegnarla continuamente, ma la stessa non può essere manipolata o cancellata selettivamente; l'intera immagine deve essere ridisegnata, quando se ne debba modificare anche solo una porzione; quindi il DVST non risulta adatto per progettazione interattiva. Il sistema raster scan conversion converte un vettore in coordinate X, Y che vengono poi visualizzate da un pennello elettronico che scandisce con continuità lo schermo (come in una televisione). Questo sistema permette di utilizzare tubi a raggi catodici di tipo standard. Peraltro, la complessità dei circuiti elettronici di supporto limitano la sua popolarità. La tecnologia di visualizzazione a raster scan (scansione a trama) usa l'approccio classico usato nella rappresentazione televisiva delle immagini ed è stata la tecnologia di visualizzazione dominante in gran parte dei terminali alfanumerici.

Per applicazioni grafiche si fa uso, invece, di un approccio di tipo bit-mapped. In un sistema bit-mapped l'immagine è composta da una matrice di "elementi" (punti) dell'immagine (pixel) che vengono memorizzati in

W. Miller della AMD Sunnyvale (CA)

Tabella 1 - Confronto fra differenti tecnologie di visualizzazione

	VANTAGGI	SVANTAGGI
Sistema STROKE	* Alta risoluzione * Elettronica di generazione del video di tipo economico * Rapido aggiornamento dell'immagine	* Circuito di deflessione e tubo costosi. * Necessità di rintracciare rapidamente l'immagine (sfarfallìo in rapporto alla complessità della immagine). * DVST (cancellazione non selettiva).
Sistema RASTER SCAN CONVERSION	* Costi più bassi rispetto al sistema stroke.	* Circuiti di scan conversion complicati.
Sistema RASTER SCAN BIT-MAPPED	* Approccio simile a quello di un normale televisore. * Usato per anni nei display alfanumerici. * Può produrre grafica bit-mapped. * Può produrre immagini complesse prive di sfarfallìo.	* Risoluzione inferiore rispetto al sistema stroke. * molta memoria coinvolta

una memoria di display (comunemente nota come frame buffer). L'immagine viene rinfrescata, linea per linea, illuminando o meno ogni volta sul display l'elemento puntuale sul quale è, nell'istante in questione, posizionato il pennello elettronico.

Nel passato il costo di RAM ad alta densità ha limitato l'ampio utilizzo di grafica bit-mapped. La Tabella 2 mostra la drastica riduzione nella quantità di componenti di memoria richiesti per i vari tipi di display in bianco e nero ottenuti utilizzando memorie ad alta densità. Per immagini a colori i numeri riportati in Tabella 2 devono essere incrementati di un fattore 3 o 4; perciò, un display a colori di 1024 x 1024 punti, mentre richiedeva migliaia di chip nei primi anni settanta, oggi ne richiede solo qualche decina. Un vantaggio del metodo raster scan bit-mapped è quello che il ritmo di rinfresco dell'immagine è indipendente dalla complessità della stessa.

La progettazione interattiva con tecnologia bit-mapped

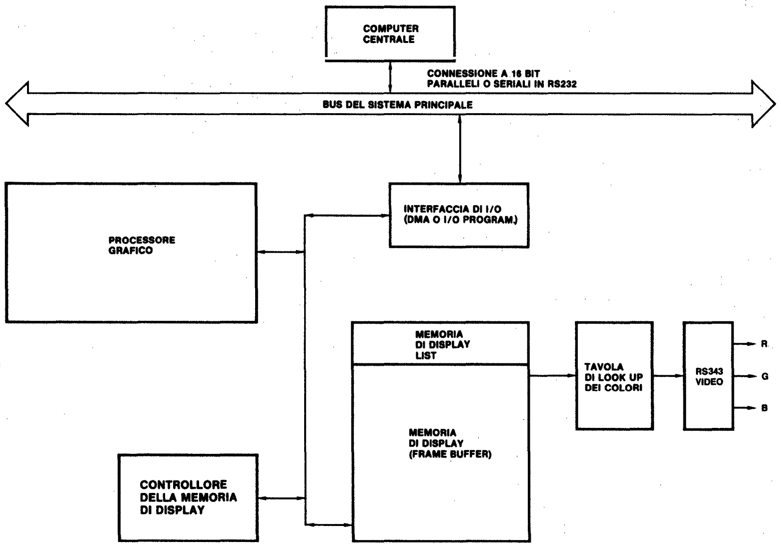
L'applicazione di display bit-mapped va dalla grafica commerciale, a stazioni di lavoro per ufficio, ad elaborazioned'immagine per CAD/CAM. Le prestazioni richieste da un sistema grafico si identificano principalmente con la risoluzione richiesta e il grado di interattività che si vuole raggiungere. In altre parole, la dinamica di aggiornamento e la dinamica di movimento, in risposta alle richieste dell'utente. La dinamica di aggiornamento è la capacità di cambiare colore, forma e proporzioni dell'oggetto raffigurato. La dinamica di movimento è la capacità di cambiare prospettiva tra l'oggetto e l'osservatore, magari ruotando il primo o ribaltandolo, ecc. Un sistema a grafico bit-mapped deve gestire una *grande area di memoria*, specialmente quando si fa uso del colore. Un display con una risoluzione spaziale di 1024 x 1024 punti e 16 colori richiede una memoria di 4 Mbit. Il frame buffer è spesso più ampio per consentire la memorizzazione addizionale di caratteri diversi (serie diverse di caratteri tipografici) e display list che contengano i comandi provenienti dal computer di controllo.

In un'immagine 1024 x 1024, rinfrescata 30 volte ogni secondo, il pixel deve essere letto dal buffer e presentato sul CRT, tramite circuiti di elaborazione video, ogni 24 nano-secondi.

Per avere una modalità di utilizzo interattiva è necessario poter aggiornare rapidamente il display, anche per operazioni concettualmente abbastanza semplici. Per esempio, un display per office automation potrebbe contenere riquadri per testi differenti; questi riquadri possono essere fatti scorrere individualmente, possono sovrapporsi e possono essere spostati; in qualunque posizione dello schermo. Aggiornare un display di questo tipo richiede la manipolazione di centinaia di migliaia di pixel nel tempo di aggiornamento del quadro. Il limite massimo viene calcolato considerando la larghezza di banda necessaria per aggiornare l'intero schermo nel tempo di permanenza di un quadro o

Tabella 2 - Requisiti di memoria per un sistema bit-mapped

Dimensioni della memoria					
Dimensioni dello schermo	**Bit**	**1K**	**4K**	**16K**	**64K**
256 x 256	64K	64K	16K	4K	1K
512x512	256K	256K	64K	16K	4K
1024x1024	1M	1K	256K	64K	16K


COMPUTER CENTRALE
CONNESSIONE A 16 BIT PARALLELI O SERIALI IN RS232
BUS DEL SISTEMA PRINCIPALE
PROCESSORE GRAFICO
INTERFACCIA DI I/O (DMA O I/O PROGRAM.)
MEMORIA DI DISPLAY LIST
MEMORIA DI DISPLAY (FRAME BUFFER)
TAVOLA DI LOOK UP DEI COLORI
RS343 VIDEO
R
G
B
CONTROLLORE DELLA MEMORIA DI DISPLAY
Fig. 1 - Architettura di un controllore intelligente per grafica bit-mapped.

"fotogramma" dell'immagine. Lo scorrimento ordinato di uno schermo di 1024 x 1024 punti richiede una larghezza di banda di 60 Mbit/secondo perchè sia la funzione di lettura che quella di scrittura devono essere effettuate per tutti i pixel del quadro entro 1/30 di secondo. Perciò, un sistema grafico interattivo bit-mapped necessita della capacità di muovere i dati con velocità elevatissime. Questi problemi vengono risolti utilizzando un processore grafico ad alte prestazioni come quello mostrato in Fig. 2 scaricando, in tal modo, il computer centrale di alcune lunghe operazioni grafiche. I comandi, sotto forma di unità elementari di programma, vengono inviati dal computer centrale al processore grafico tramite ad memoria che contiene le display list. Essi vengono, poi interpretati dal processore grafico affinchè vengano eseguite le funzioni desiderate. Interazione in tempo reale significa anche che il processore grafico deve avere abbastanza intelligenza per decodificare una sequenza di azione (aggiornamento/dinamica di movimento) senza molto intervento da parte dell'elaboratore principale. La memoria di display, chiamata anche frame buffer (o buffer di quadro), come si è detto, è tipicamente una memoria tridimensionale dove le coordinate X,Y sono la risoluzione (per esempio 512 x 512 o 1024 x 1024) e le coordinate Z contengono le intensità o colori (per esempio un frame buffer a 4 od 8 livelli permette la visualizzazione simultanea di rispettivamente 16 o 256 colori). Inoltre, le uscite del frame buffer vengono interpretate tramite una tabella che associ ad ogni codice uno fra i colori previsti nella "tavolozza" del sistema. Per esempio una tabella a tre canali, ciascuno con uscite ad 8 bit, permette l'utilizzo di una tavolozza di 16,7 milioni di colori. Le uscite della tabella dei colori vengono convertite in segnali analogici così da pilotare i pennelli colore del video. Il display memory controller si collega alla frame memory e svolge tre funzioni: rinfresco del video, aggiornamento del video e rinfresco della memoria dinamica. Il rinfresco del video richiede l'invio del contenuto della memoria di display al CRT, e velocità abbastanza elevata da evitare sfarfallìo dell'immagine. L'aggiornamento del video ha luogo durante il ritorno di quadro del pennello elettronico. Il rinfresco della memoria avviene ad intervalli regolari così da mantenere sempre valido il contenuto della RAM dinamica usata come frame buffer (buffer di quadro). Il processore grafico è il cuore del sistema. Alcune delle funzioni tipiche di un processore grafico sono:

(1) indirizzamento diretto della memoria di frame buffer così che sia possibile un rapido aggiornamento. La gestione della generazione dell'immagine e della display list dovrebbero essere eseguite interattivamente;
(2) alta capacità di movimentazione dei dati e di manipolazione dei singoli bit, per poter facilmente tracciare vettori, caratteri e riempire poligoni;
(3) riconoscimento in tempo reale di condizioni particolari quali il sincronismo orizzontale, poichè i pixel vengono modificati solo durante il ritorno di quadro per evitare problemi di sovraccarico di memoria;
(4) "intelligenza" distribuita per poter aggiornare il display interattivamente senza intervento del computer centrale;
(5) microprogrammabilità, così da permettere flessibilità nelle esecuzioni di vari macro-comandi tramite firmware;
(6) possibilità di autodiagnostica così da assicurare un funzionamento affidabile e facilità di manutenzione come richiesto dalla maggior parte dei sistemi sul mercato.

Un controllore grafico microprogrammabile, con prestazioni elevate può essere realizzato usando diversi dispositivi della famiglia Am2900, tra cui il più avanzato è il microprocessore a 16 bit Am29116.

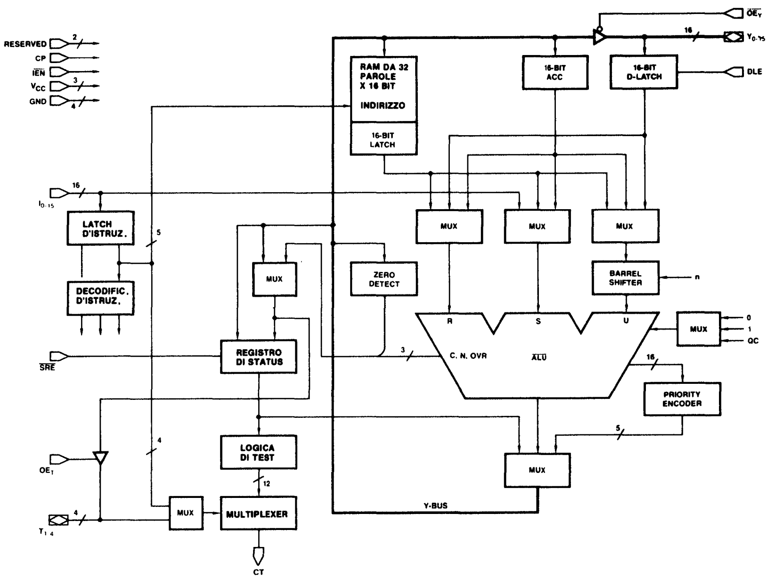


Fig. 2 - Diagramma a blocchi dell'Am29116

L'architettura dell'Am29116

L'Am29116 è un microprocessore a

16 bit potente, capace di operare con un tempo di micro-istruzione di 100 nsec.

Esso ha caratteristiche particolarmente adatte per una movimentazione rapida dei dati e per la manipolazione dei singoli bit come necessario in applicazioni grafiche bit-mapped. L'Am29116 comprende una RAM 32 x 16 sul chip con un registro tampone sugli ingressi, un accumulatore a 16 bit, un codificatore di priorità a 16 bit, un registro di stato (dove vengono memorizzate le flag di stato), un generatore-multiplexer di conditional code, 16 buffer di uscita tri-state e registro tampone e decodificatore per le istruzioni a 16 bit (Fig. 2).

La RAM single-port ha latch di uscita che sono trasparenti quando l'ingresso del clock CP è alto ed in ritenuta (latched) quando CP è bassa.

TECNOLOGIE

I dati vengono scritti nella RAM mentre il clock è basso se l'ingresso $\overline{\text{IEN}}$ è pure basso e se l'istruzione che si sta eseguendo sceglie la RAM come destinazione. I dati vengono scritti negli 8 bit inferiori della parola indirizzata. Si possono usare indirizzi separati di lettura e scrittura per la RAM impiegando un multiplexer sugli ingressi da I4 a I0 e usando CP come segnale di select. L'accumulatore accetta dati nella transizione da 0 ad 1 di CP se $\overline{\text{IEN}}$ è anch'esso zero e se l'istruzione che si sta eseguendo lo sceglie come sua destinazione. Come nelle locazioni di memoria della RAM, le istruzioni di un byte modificano solo la metà inferiore dell'accumulatore, mentre le istruzioni di una parola (16 bit) modificano l'intero registro. Il latch d'ingresso dei dati mantiene i dati in ingresso alla ALU (Arithmetic Logic Unit) sul bus bidirezionale Y.
Le sorgenti di dati per la ALU sono la RAM, l'accumulatore e il latch di ingresso. Il barrel shifter a 16 bit è uno degli ingressi della ALU. Questo permette la rotazione dei dati fino a 15 posizioni sia che si tratti di byte (8 bit) o 16 bit. La ALU, che può operare su uno, due o tre operandi a seconda dell'istruzione che si sta eseguendo, contiene un circuito di lockahead carry generation per tutti i 16 bit. Tutte le operazioni della ALU possono essere effettuate su 8 o 16 bit. Lo status output Carry (C), Negative (N) ed Overflow (OVR - condizione in cui il risultato di un'operazione aritmetica supera la capacità di un registro) vengono generati sia a livello di 8 che di 16 bit. Una quarta flag, Zero (Z), viene generata all'esterno della ALU ed anch'essa può operare su 8 o 16 bit. Il bit di Stored Carry (QC) dello status register può essere selezionato come ingresso di carry della ALU per favorire operazioni aritmetiche multi-precision. Il Codificatore di Priorità (priority encoder) produce un codice binario per indicare la locazione dell'1 di ordine maggiore al suo ingresso. L'ingresso del Codificatore di Priorità viene generato dalla ALU che svolge anche operazioni di AND sull'operando su cui si deve stabilire la priorità e su una maschera. Nelle operazioni a 16 bit, se nessun bit è a 1, l'uscita del codificatore di priorità è uno 0. Se il bit 15 è a 1 l'uscita è un 1 binario. Il bit 14 produce un 2 binario, ecc. Alla fine se soltanto il bit 0 è a 1 il risultato è un 16 binario. Lo status register ad 8 bit ed il generatore/multiplexer di condition code contengono le informazioni e la logica necessari a sviluppare 12 segnali

di test per condition code. Il multiplexer sceglie un segnale di test e lo pone sull'uscita condition code test (CT) così che possa essere usato dal sequenziatore del microprogramma. Il multiplexer può essere indirizzato in due modi; nel primo, che è usato per ottenere il massimo rendimento, il T-bus viene usato solo per input per specificare direttamente la posizione di selezione del multiplexer.

Nel secondo, l'uscita CT viene selezionata attraverso un'istruzione di test. L'ingresso dell'output-enable Y-bus ($\overline{OEY}$) abilita i buffer tri-state di uscita quando si trova a livello basso.

Quando $\overline{OEY}$ è alto, i buffer di uscita vengono posti nella condizione di alta impedenza (permettendo così che i dati vengano inviati al D-latch dal data bus a 16 bit del controllore).

Il registro tampone d'istruzione a 16 bit è normalmente trasparente per consentire la codifica dei 16 ingressi di istruzione I/15-0 in segnali di controllo interni per l'Am29116 e l'esecuzione delle istruzioni in un unico ciclo di clock. Le uniche eccezioni a questa regola sono le istruzioni di immediate-operando che vengono eseguite in due cicli di clock anzichè uno. Queste vengono memorizzate nel latch delle istruzioni nel primo ciclo di clock ed eseguite durante il secondo. È durante il secondo ciclo che l'operando-immediato, residente nel campo I/15 0 del la prossima macroistruzione, viene portato in esecuzione e quest'ultima giunge a compimento. Le istruzioni immediate sono utili ogni volta che si utilizzino maschere e costanti speciali. (Lo Am29116 permette l'addizione o la sottrazione di 2^N e l'uso di 2^N, e del suo complemento, come maschera, per qualsiasi valore di N compreso fra 0 e 15, in un solo ciclo di clock).

Le istruzioni dell'Am29116

Le istruzioni a 16 bit dell'Am29116 possono essere raggruppate in undici tipi differenti che corrispondono in modo naturale alla logica di decodifica interna: operando singolo, traslazione di bit singolo (shift), operando doppio, rotate ad merge, bit oriented; ruotare per n bit, ruotare e comparare, "prioritizzare", verifica ciclica della ridondanza (redundancy check): una verifica che richiede la presenza di caratteri o bit extra per consentire l'individuazione automatica degli errori. I caratteri addizionali non contribuiscono in se all'informazione e vengono, perciò, descritti come ridondanti, sebbene essi abbiano una loro funzione importante. La ridondanza in generale è l'uso di più unità di quanto sia necessario, così garantendo un elemento di sicurezza in caso di guasto dell'elemento principale; status e no-op (ovvero no operation). Un file per sistema 29 AMDASM DEF è disponibile per fornire all'utente mnemonics predefiniti per microcodificare l'Am 29116. Per esempio, la file line AMDASM SRC:

SOR W, INC, SORY, R1

incrementa l'intero contenuto a 16 bit di una locazione di memoria RAM numero 1 dell'Am29116 di uno e la pone sul bus Y, e

TOR B, SUBR, TORAR, R2

sottrae il byte di ordine inferiore dell'accumulatore dal byte di ordine inferiore della locazione di memoria RAM numero 2, mentre lascia immutato il byte superiore della stessa locazione.

La Tabella 3 riassume le operazioni base che le istruzioni dell'Am29116 possono eseguire in un singolo ciclo. (Due cicli vengono usati se un operando è composto da dati immediati). Si noti che per una linea tipica di questa tabella, ci sono diversi codici operativi mnemonici, a seconda della scelta di sorgente e destinazione degli operandi. Molte di queste operazioni si dimostrano particolarmente utili quando si usa un controllore grafico "intelligente".

Per esempio, la manipolazione di una lunghezza arbitraria e dei campi dati dei pixel, allineati in maniera arbitraria, viene facilitata dall'istruzione rotate and merge. Un operando a 16 bit viene ruotato di N bit e combinato con un secondo operando a 16 bit sotto maschera, entro un singolo ciclo. Le istruzioni che addizionano e sottraggono 2^N e caricano in memoria 2^N ed il suo complemento, permettono la creazione di maschere di uso comune in modo semplice ed efficiente. Le operazioni di video sottoposto ad inversione ed immagine sottoposta ad OR esclusivo con sfondo richiedono istruzioni di bit level set, reset e test. L'individuare i bordi di un poligono da riempire richiede istruzioni di "prioritizzazione".

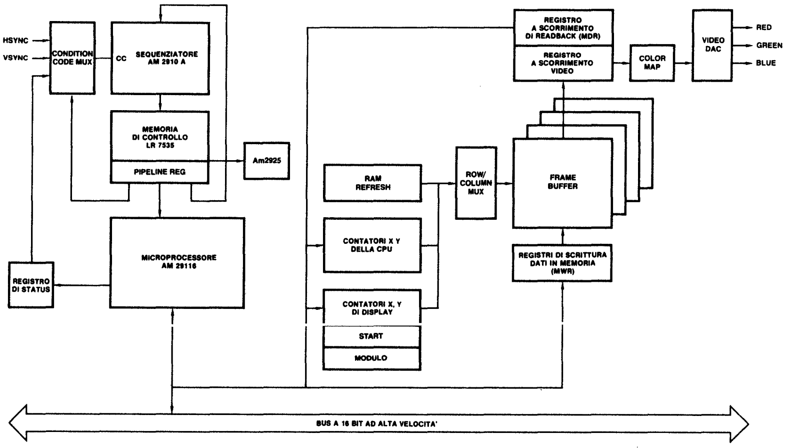


Fig. 3 - Diagramma a blocchi semplificato del processore grafico bit-mapped.

Tabella 3 - Operazioni a singolo ciclo di clock dell'Am29116

ADD
Add with Carry
Add 2^N
And
Complement
Accumulate forward CRC
Accumulate reverse CRC
Exclusive Nor
Exclusive Or
Increment
Load 2^N
Load 2^N Complemented
Move
Nand
Negate (2's complement)
Nor
Or
Prioritize under mask
Reset bit N
Reset status bit
Rotate N bit
Rotate N bit and compare under mask
Rotate N bit and merge according to mask
Set bit N
Set status bit
Single bit shift
Subtract
Subtract with Carry
Subtract 2^N
Test bit N
Test status bit

L'architettura di un sistema bit-mapped basato sull'Am29116

La Fig. 3 mostra un diagramma a blocchi semplificato del processore grafico bit-mapped basato sull'Am29116. Fornisce tutte le funzioni e caratteristiche discusse in precedenza. L'Am29116 fornisce la capacità di tracciare unità base di programma grafico nel frame buffer ad alta velocità e col minimo coinvolgimento del computer centrale. Il processore può tracciare vettori, caratteri e riempire poligoni alla velocità di un pixel ogni 100 nano secondi; con una velocità media di tracciatura di 2 milioni di pixel per secondo poichè essi possono essere tracciati unicamente durante il ritorno di quadro. Il processore grafico microprogrammato ha una sezione di controllo ed una sezione di passaggio dei dati. La sezione di controllo consiste di un sequenzializzatore di microprogramma (Am2910A), per produrre il prossimo indirizzo di programma che deve essere registrato nella memoria di controllo (Am29S35). Le flag di stato della ALU, il sincronismo orizzontale ed altre condizioni di test del sistema vengono esaminate dal sequenzializzatore per determinare il flusso di programma. I macro-comandi vengono immagazzinati in qualche tipo di memoria di display list e vengono interpretati dal sequenzializzatore.

Questi comandi vengono acquisiti dalla CPU centrale attraverso un I/O programmato oppure un'interfaccia tipo DMA (direct memory access) e facilmente realizzati tramite il generatore d'indirizzo DMA Am2940 ed il data port parallelo di I/O Am2950. Il controllore del display consiste di tre sezioni: rinfresco della memoria dinamica, aggiornamento del video e refresh del video. L'aggiornamento del video avviene durante il ritorno di quadro e viene eseguito scrivendo pixel nelle locazioni di memoria indicate dai contatori X ed Y della CPU. Esiste anche un registro di memorizzazione dei dati scritti in memoria (Memory Write data Register-MWR) per l'Am29116, così che quest'ultimo non debba aspettare un completo ciclo di memoria durante l'operazione di scrittura. Il rinfresco del video avviene tramite i contatori di display X ed Y. I dati del frame buffer vengono trasferiti in un registro a scorrimento dedicato alla funzione video ed in congiunzione alla look-up table dei colori è possibile visualizzare 16 colori contemporaneamente da una tavolozza di 16,7 milioni di tinte. Esiste pure un percorso per il read-back dei dati nel frame buffer eseguito dall'Am29116 (il read-back check è una verifica dell'accuratezza nella trasmissione di dati l'informazione trasmessa viene inviata nuovamente alla sorgente e paragonata all'informazione originale).

L'architettura mostrata utilizza un Am29116 per controllare livelli multipli di memoria. C'è un registro di mascheratura alla scrittura che funziona assieme ai circuiti di pattern nel modificare selettivamente ogni livello di memoria. L'Am29116 è il cuore del sistema che fornisce le capacità per produrre una grafica a colori ad alta risoluzione ed alte prestazioni, attraverso la sua architettura controller oriented ed il suo set d'istruzioni ottimizzato per la movimentazione dei dati e la manipolazione dei bit.

Operazioni grafiche

L'architettura dell'Am29116 è ideale per utilizzare le operazioni grafiche primitive necessarie in un processore a grafica bit-mapped. Un esempio di set d'istruzioni per un processore grafico viene fornito in Tabella 4. La maggior parte delle operazioni fanno riferimento a due registri di pointer P1 e P2, residenti nel processore grafico. Per esempio il comando MOV_{P1} X, Y caricherà P1 con le locazioni specificate da X ed Y.

L'istruzione DRAW traccerà una linea tra P1 e P2. L'uso di questi due registri dedicati riduce il numero di bit necessari a definire un'istruzione. Questo fa sì che il processore grafico possa essere facilmente interfacciato a qualsiasi bus di microprocessore a 16 bit, sia per mezzo di una porta di I/O che attraverso un'operazione di DMA.

Definire un punto

L'operazione grafica più primitiva visualizza un punto sul display. Questa azione primitiva, quando ripetuta, può essere utilizzata per costruire strutture più complesse. Strutture grafiche come linee, rettangoli e poligoni possono essere costruiti partendo da punti singoli, similmente a quanto fatto con la tecnica delle stampanti a matrici di punti. In aggiunta, alcuni degli algoritmi più complicati richiedono che si possano aggiungere o togliere punti individuali. Per esempio, le tecniche di affinamento delle linee richiedono la possibilità di aggiungere punti individuali adiacentemente ad alcune linee per minimizzare il loro aspetto frastagliato. L'algoritmo per tracciare un punto sul display è un'operazione di traduzione di un singolo indirizzo. Al processore di display viene fornito un punto sotto forma di coordinata assoluta (è anche possibile avere indirizzi relativi ai registri di cursore P1 e P2 e li si realizza facilmente elaborando un indirizzo assoluto nella memoria del display e il processore deve mappare quella locazione di bit nella

Tabella 4 - Set d'istruzioni del controllore di display

1.	**Inizializzazione**	
	INIT	reset ed inizializza
2.	**Movimentazione del cursore di display**	
	MOVP1 x,y	muovi il cursore (pointer) 1 ad x,y
	MOVP2 x,y	muovi il cursore 2 and x,y
	RMOVP1 dx,dy	muovi relativamente il cursore 1 in funzione di dx, dy
	RMOVP2 dx,dy	muovi relativamente il cursore 2 in funzione di dx,dy
	POLYS	inizia a definire un poligono
	POLYV x,y	aggiungi il vertice del poligono
3.	**Comandi di tracciatura**	
	DRAW	traccia un vettore
	CHAR	traccia un carattere in P1
	ARC n	traccia un arco o cerchio usando P1 come centro e parti dal punto P2; la lunghezza dell'arco o la circonferenza del cerchio è di n pixel.
	RECT-1	traccia un rettangolo definito da P1 e P2
	RECT-2	riempi il rettangolo definito da P1 e P2
	FFILL	riempimento lampeggiante del rettangolo definito da P1 e P2
	CLEAR	azzera la memoria d'immagine
	POLYF	polygon fill (riempimento del poligono)
	POLYO	definisci il contorno del poligono
	AFILL-1	riempimento di un'area a caso a partire da un punto base o di riferimento.
	ALFILL n	riempi per la lunghezza di un certo percorso a partire da P1.
4.	**Comandi di controllo della tracciatura**	
	SETD d	set di registro dei dati di tracciatura.
	SETM d	set di registro di mascheratura per la tracciatura (mask register).
	PATT d	seleziona il modo ed il modello di tracciatura.
	SECT-1 d	seleziona la dimensione dei caratteri.
	SECT-2 d	seleziona la rotazione del carattere ed il character mirror
5.	**Comandi di controllo del display**	
	ZOOM d	seleziona i fattori di zoom orizzontali e verticali
	PPAN x,y	pan all'origine del punto x,y
	CMAP a,d	carica l'indirizzo del color map con i dati.
	BLNKON	metti in funzione il modo lampeggiante
	BLNKOF	disinserisci il modo lampeggiante.
6.	**Comandi di controllo del trasferimento dati**	
	WRR d ...	leggi in blocco il rettangolo definito da P1 e P2
	RDR	leggi in blocco il rettangolo definito da P1 e P2
	PIXBLT w, h, f	muovi in blocco i pixel del rettangolo di ampiezza w, altezza h e direzione f da P1 a P2
	READP	leggi il colore del pixel in P1
	WRITEP c	scrivi un pixel in P1 col colore definito
7.	**Comandi di cursore**	
	CURS	abilita la visualizzazione del cursore nel punto 1
8.	**Comandi di servizio**	
	CRCRD	leggi il registro dei dati CRC
	SYNCH	aspetta il ritorno verticale del pennello elettronico (per l'animazione dell'immagine)
	READC	leggere la configurazione hardware e la versione microcode.

Per gentile concessione della Metheus

map memory. Questo mappaggio da indirizzo logico ad indirizzo fisico è necessario perchè la memoria logica di display di 1K x 1K viene in effetti realizzata con 16 bit di chip di memoria da 64K. La Fig. 4 mostra l'organizzazione delle due memorie, ed i diversi formati d'indirizzo. L'operazione di traduzione semplicemente richiede due indirizzi a 10 bit e genera un indirizzo per parola a 16 bit ed un indirizzo per bit a 4 bit. Come mostrato in Fig. 5 l'istruzione di "ruotare e fondere" può essere usata per costruire la nuova parola a 16 bit in due cicli.

Questo dà come risultato la quantità desiderata di 16 bit. Il compito rimanente di fissare un bit singolo nella parola a 16 bit può essere facilmente realizzato leggendo la parola dalla memoria di display ed usando l'istruzione 2^N prefissata, con la quantità a 4 bit X_1 come index (sia attraverso una tabella di jump, o tramite un registro ad N bit).

La parola modificata viene poi riscritta nella memoria di display col risultato di ottenere sul display il bit voluto. Realizzazioni alternative producono il mapping con l'hardware,

permettere la definizione di oggetti di alta complessità. Poligoni ed altri oggetti più complicati possono essere costruiti partendo da questa istruzione primitiva. L'istruzione draw traccerà una linea tra due punti (P1 e P2 in Fig. 6) nella memoria del display. Due casi speciali esistono quando P1 e P2 risulteranno essere su una linea orizzontale o verticale (Fig. 6). Questi casi possono essere affrontati verificando se $Y_1=Y_2$ e $X_1=X_2$ e richiamando delle routine particolari per gestire le linee orizzontali o verticali. Come si vede in Fig. 8 una linea orizzontale si può pre-

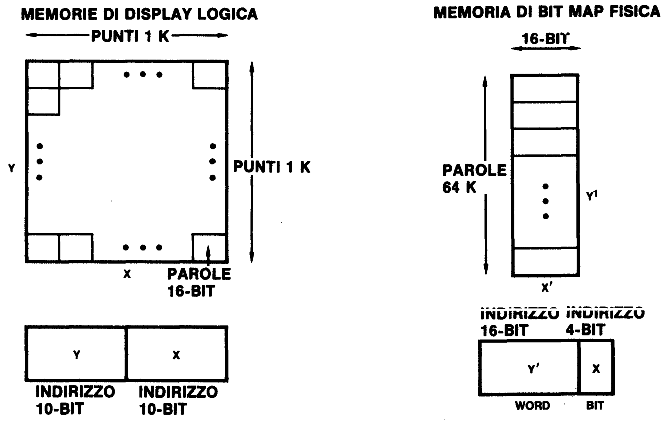


Fig. 4 - Organizzazione della memoria e traduzione dell'indirizzo

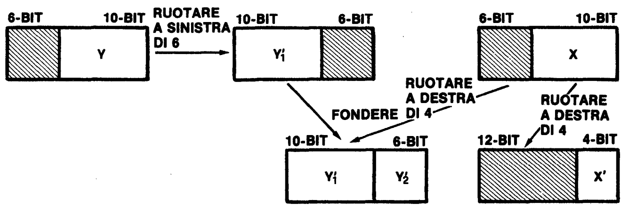


Fig. 5 - Esemplificazione dell'istruzione "ruotare e fondere".

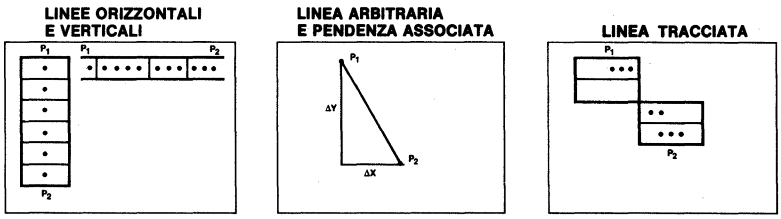

LINEE ORIZZONTALI E VERTICALI
P1
P1
P2
P2
LINEA ARBITRARIA E PENDENZA ASSOCIATA
P1
ΔY
ΔX
P2
LINEA TRACCIATA
P1
P2

Fig. 6 - Istruzione DRAWN

Il primo ciclo fornisce il margine sinistro dell'indirizzo Y a 10 bit ruotandolo verso sinistra di 6 posizioni di bit. I 6 bit superiori dell'indirizzo X possono essere ruotati verso destra e inseriti nelle sei posizioni lasciate vacanti dall'operazione precedente.

ma questo sarebbe più difficile da modificare nel momento in cui si vogliano accrescere le dimensioni della memoria. La prossima operazione primitiva di cui dobbiamo discutere è DRAW. Questa operazione congiunge punti con una linea o serie di linee così da

sentare in due casi separati; le due estremità contenute nella stessa parola a 16 bit, oppure memorizzate separatamente con meno parole frapposte tra una e l'altra. Nel primo caso i bit tra P1 e P2 devono essere determinati. Questo può essere realizzato nell'Am29116 stabilendo per prima cosa tutti i bit sopra P1 e poi stabilendo i bit sopra P2. Queste operazioni possono essere realizzate facilmente nell'Am 29116 generando una parola maschera per consentire la determinazione o la ri-determinazione dei bit superiori od inferiori di una parola. La linea orizzontale con più parole può essere manipolata usando il set di istruzioni sulla parola inferiore, determinando selettivamente tutte le parole intere esistenti fra P1 e P2 (se ce ne sono), ed alla fine determinando tutti i bit inferiori a P2. Le linee verticali sono molto facili da trattare. I 4 bit inferiori dell'indirizzo specificano quale bit, nella parola a 16 bit, deve essere posto a "1". Questa operazione viene, poi, ripetuta sull'intero campo esistente fra Y1 ed Y2. L'operazione di set 2^N nell'Am29116 è ideale. Una situazione più complicata si può verificare quando P1 e P2 non sono orizzontali o verticali. In questa situazione una pendenza esiste fra P1 e P2 (rapporto Y/X). Questa pendenza viene usata per calcolare la successione di punti necessari a tracciare la linea. Il calcolo della pendenza richiede un'operazione di divisione, un divisore a 10 bit ed un dividendo a 10 bit con un quoziente a 20 bit. Una volta che la pendenza viene calcolata l'algoritmo di tracciatura della linea usa operazioni di somma successive per computare la locazione dei punti. La locazione X viene incrementata di 1 e la locazione di Y dalla pendenza (il calcolo della locazione Y è un calcolo a 20 bit per consentire di operare anche nel peggior caso di pendenza, uguale a 1/1024). Fino a che i 10 bit superiori del contatore Y non cambiano, il punto alla locazione X è determinato. Quando i 10 bit superiori del contatore Y incrementano la parola viene scritta nella memoria e si va a prendere la parola seguente. Questo algoritmo termina quando P2 è stato determinato. L'Am29116 contiene tutte le risorse ed istruzioni necessarie a realizzare l'algoritmo di tracciatura della linea. La divisione di pendenza (slope) può essere realizzata nell'Am29116 come una successione di operazioni di sottrazione e scorrimento (shift). Con 4 cicli per bit e 20 bit questo richiederà circa 8 cicli (a 100 nsec per ciclo, ciò equivale a solo 8 microsecondi). Poichè la pendenza deve essere calcolata solo una volta per linea, si ha un basso costo intrinseco di questa funzione.

Infatti, questo calcolo può essere sovrapposto al tempo di accesso della memoria e risultare quindi come inesistente in termini di perdita di tempo. Il contatore a 20 bit può essere realizzato con 2 dei 32 registri a 16 bit per memoria scratch pad. L'istruzione "incrementa con resto" viene usata per ottenere un indirizzo di 20 bit. Sono possibili istruzioni di DRAW più complicate.

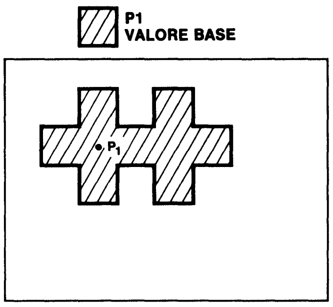


Fig. 7 - Operazione di POLYGON FILL

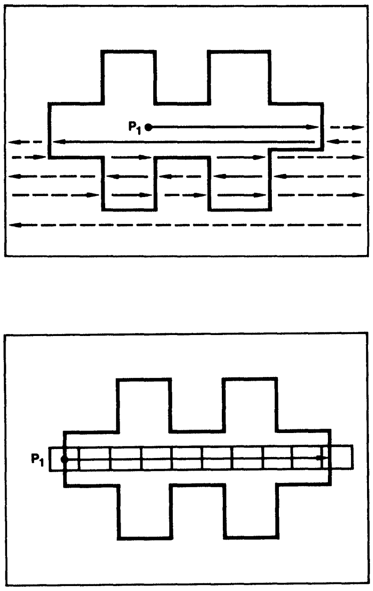


Fig. 8 - Algoritmo di POLYGON FILL

Operazioni di riempimento

Una delle funzioni grafiche più complicate di cui discuteremo è l'operazione di FILL. Questa operazione viene usata per sostituire la parte interna di una figura chiusa da un contorno con un nuovo valore. Descriveremo il caso più semplice in cui il nuovo valore è costante, ma una facile estensione sarebbe quella di permettere l'uso di qualsiasi valore arbitrario (persino un'area del display attuale) come valore di riempimento (valore di FILL).

La Fig. 7 illustra come il comando di FILL, FILL POLYGON, funzionerebbe. FILL POLYGON(P1) si sostituirebbe al POLIGONO che racchiude P1 come valore base (o di riferimento). Questa operazione viene usata il più delle volte per "mettere in evidenza una particolare struttura del display. Per esempio, in un sistema di CAD per VLSI un poligono potrebbe rappresentare un particolare contorno di diffusione o strato di metallizzazione. Aree differenti necessitano contorni o colori differenti per far sì che il progettista distingua facilmente una struttura dall'altra. Non serve altro che specificare un punto all'interno del poligono per il comando di FILL POLYGON. L'algoritmo è abbastanza sofisticato da elaborare da sè tutti i

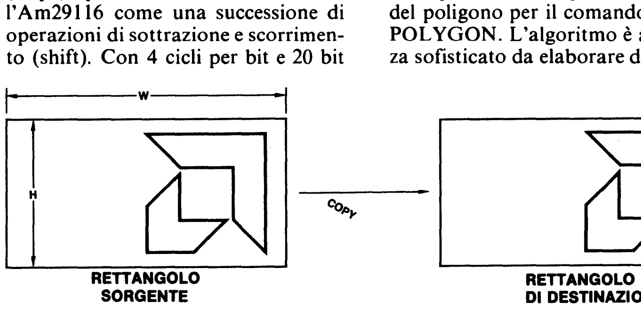


Fig. 9 - Operazione BITBLT

punti interni (o inversamente tutti i punti esterni) al poligono. Il vantaggi di spostare questa "intelligenza" nel processore grafico è una riduzione nell'overhead del computer centrale. È vero che il computer centrale contiene di per sé tutta l'informazione logica relativa al poligono (vertici, segmenti, ecc.) nel data-base della figura. L'individuazione di tutti i punti interni (od esterni) sarebbe possibile partendo da queste informazioni, ma il processore grafico può realizzare questa funzione molto più rapidamente avendo accesso ai dati non elaborati ed elaborandoli. Questa è la ragione principale per la migrazione di "intelligenza" verso la periferia del processore centrale. In molti casi una piccola quantità di "intelligenza" posta nel punto ove esistono i dati originali può drasticamente ridurre l'overhead del computer centrale.

Algoritmo di riempimento del poligono

Come illustrato in Fig. 8, l'algoritmo consiste nell'effettuare una scansione dal punto base di riferimento (P1) verso il margine del display (oppure individuare un rettangolo che contenga il poligono, delimitato da X ed Y nella Fig. 8). Ogni volta che una linea viene incrociata, la flag "interna" commuta il suo stato, indicando un cambiamento da interno a esterno e viceversa.

Questo processo viene terminato quando una linea non contiene alcuna porzione che sia "interna" al poligono in questione. Una volta che il "fondo" è stato individuato viene generata una scansione da P1 verso l'alto, ancora fino a che una linea "vuota" viene incontrata.

Queste due scansioni risultano in un poligono completamente riempito. L'Am29116 contiene due istruzioni speciali che rendono facile realizzare l'operazione FILL POLYGON. L'istruzione di priority encode (codifica della priorità) e l'istruzione di N bit set si combinano per realizzare un'istruzione primitiva set-to-boundary (fissa al contorno del poligono) che può essere usata per costruire l'istruzione FILL POLYGON. L'istruzione set-to-boundary prende un punto di partenza e fissa o determina tutti gli elementi alla destra del punto (o alla sinistra, a seconda della direzione specificata) fino a che si individua un margine. Le parole vengono fornite in incrementi di 16 bit (come visto in Fig. 11). L'istruzione di priority encode fornirà un valore associato alla locazione del primo bit non-zero della parola. Se non viene individuato alcun 1 l'intera parola viene scritta nella memoria di display. La prossima parola viene letta ed il processo viene ripetuto fino a che si scopre un 1. Quando si scopre un 1, l'uscita del priority-encoder indica l'indirizzo del bit. L'istruzione di set può allora essere usata per porre tutti i bit nella stessa condizione del bit indirizzato. Una volta che un punto di perimetro viene individuato la flag di interno/esterno viene commutata.

Quando si va fuori dal poligono un'istruzione di search-to-boundary viene impiegata per rintracciare il contorno più vicino del poligono. Questa istruzione è una semplice modifica dell'istruzione set-to-boundary; sostanzialmente l'operazione di set è stata cancellata.

Bit Block Transfer (BITBLT)

Una potente operazione base usata comunemente nella grafica coinvolge la manipolazione di una matrice (array) rettangolare di bit.

Tipicamente, l'operazione di bit block transfer (BITBLT) ha accesso ai bit di un rettangolo sorgente, ai bit di un rettangolo di destinazione, elabora una funzione di questi, e memorizza il risultato nel rettangolo di destinazione. La Fig. 9 illustra una possibile configurazione del rettangolo di sorgente e di destinazione, inseriti in due separate bit-map. Coppie successive di bit vengono prese dal gruppo di bit di sorgente e di destinazione. L'operazione viene ripetuta su tutte le tessere (pixel) corrispondenti di un'ampiezza W, e ripetuta ancora linea per linea per l'altezza H. L'operazione BITBLT può essere impiegata per effettuare varie funzioni. Per esempio, la memoria di frame buffer è generalmente più ampia dello schermo visibile e le parti invisibili solitamente contengono biblioteche di stili di caratteri diversi, menu e simboli vari. I testi e simboli individuali possono essere visualizzati specificandoli come il rettangolo sorgente per l'operazione di BITBLT e mossi in qualsiasi parte dello schermo. In modo simile, finestre sullo schermo possono essere fatte scorrere senza problemi ricopiandole nella nuova posizione e pulendo l'area liberatasi. Per mettere in opera efficacemente la BITBLT, bisogna aver accesso a bit paralleli di dati dalla memoria di frame e bisogna poterveli memorizzare. Tuttavia, i rettangoli di sorgente e destinazione possono essere posti ovunque all'interno del frame buffer e spes-

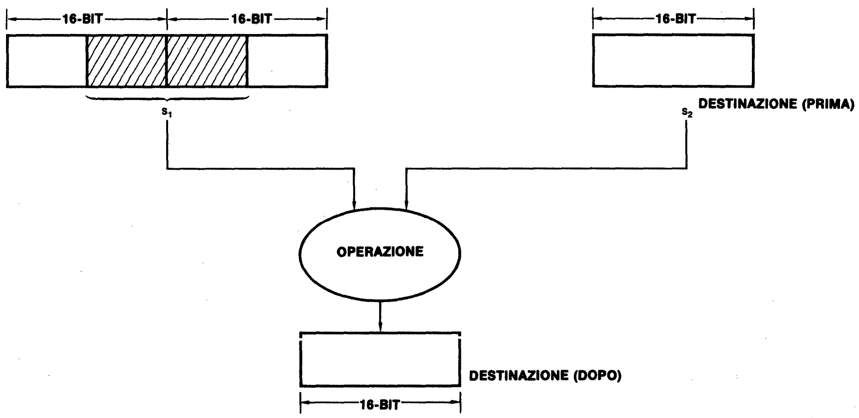


Fig. 10 - Loop interno modificato di BITBLT

so questi non sono localizzati convenientemente lungo i confini arbitrari delle parole di memoria. Se è necessario scrivere 16 pixel nella memoria di frame buffer che non siano allineati con i suddetti confini, l'istruzione di write potrebbe dover essere spezzata in due "write" separati per due parole di memoria adiacenti. Il loop interno di BITBLT legge 16 bit dai rettangoli di sorgente e di destinazione. Nessuno di questi due rettangoli deve essere allineato in termini di parole di memoria. Tuttavia, gestendo le condizioni finali appropriatamente, il loop interno può essere modificato così che la parola di destinazione sia allineata coi confini arbitrari dell'organizzazione della memoria, per quanto la parola sorgente possa trovarsi a metà fra due parole differenti come mostrato in Fig. 10. 16 bit vengono estratti da due parole adiacenti del rettangolo sorgente e combinati con la parola di destinazione così da formare il risultato. Le due parole di sorgente devono essere ruotate per consentire l'estrazione dei 16 bit desiderati. Il conto di quale sia la misura di cui devono essere ruotate viene dato dalla differenza d'allineamento tra il rettangolo sorgente e quello di destinazione. Le due parole ruotate vengono poi mascherate e combinate per produrre la parola di destinazione. Il conto di rotazione e la maschera necessaria possono essere pre-computate per ciascuna operazione BITBLT. La Fig. 11 illustra una operazione di copiatura variata spesso usata nel BITBLT ed il loop interno corrispondente usato dall'Am29116. La maschera viene pre-computata e memorizzata in RAM 0 dell'Am29116.

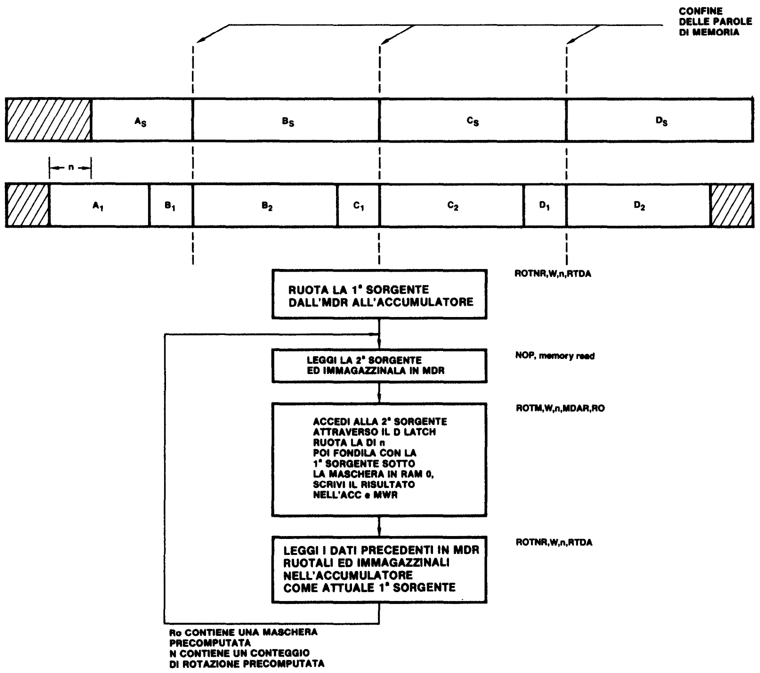

CONFINE DELLE PAROLE DI MEMORIA
A_S
B_S
C_S
D_S
n
A_1
B_1
B_2
C_1
C_2
D_1
D_2
ROTNR,W,n,RTDA
RUOTA LA 1ª SORGENTE DALL'MDR ALL'ACCUMULATORE
NOP, memory read
LEGGI LA 2ª SORGENTE ED IMMAGAZZINALA IN MDR
ROTM,W,n,MDAR,RO
ACCEDI ALLA 2ª SORGENTE ATTRAVERSO IL D LATCH RUOTA LA DI n POI FONDILA CON LA 1ª SORGENTE SOTTO LA MASCHERA IN RAM 0, SCRIVI IL RISULTATO NELL'ACC e MWR
ROTNR,W,n,RTDA
LEGGI I DATI PRECEDENTI IN MDR RUOTALI ED IMMAGAZZINALI NELL'ACCUMULATORE COME ATTUALE 1ª SORGENTE
Ro CONTIENE UNA MASCHERA PRECOMPUTATA
N CONTIENE UN CONTEGGIO DI ROTAZIONE PRECOMPUTATA

Fig. 11 - Operazione di copia del loop interno.

Il conto di rotazione, r, viene anch'esso precomputato. Può essere memorizzato in un registro-N esterno per modifica dell'istruzione "rotate". Alternativamente, il conto precalcolato di rotazione può anche essere usato come indice per una branch-table (o jump-table) con 16 istruzioni separate per ciascun conto di rotazione appropriato. Qui, la scelta è fra tempo d'esecuzione (avere un registro N-count esterno aggiungerà il ritardo, dovuto allo stadio di multiplexing, ai tempi tipici di esecuzioni dell'Am29116 e memoria di controllo. Si noti che il loop interno richiede solo tre cicli e due microistruzioni dell'Am 29116.

Tuttavia, due cicli di memoria vengono eseguiti dalla memoria di frame buffer (uno di scrittura ed uno di lettura). Si può anche usare l'Am2925 per estendere i cicli delle microistruzioni durante questi tempi. Diversamente, si può progettare una memoria di frame buffer con una maggiore ampiezza di banda passante, così che il loop intero possa procedere al massimo della velocità.

Operazioni aggiuntive

Si possono effettuare operazioni più complesse con le operazioni primitive discusse in precedenza. La rotazione di un poligono potrebbe consentire di orientare un oggetto in qualsiasi direzione. Per esempio, nella Progettazione Assitita da Calcolatore (CAD Computer Aided Design) per un'apparecchiatura meccanica, si potrebbe definire una piastra metallica, poi la si potrebbe porre in qualsiasi orientamento sulla struttura assemblata. La creazione di finestre apre una veduta sulla porzione di un diagramma differente, mostrata sul display assieme al disegno principale. In questo modo si possono avere nello stesso momento figure differenti sul display. Per esempio una finestra che mostri una veduta globale e strategica su un display tattico più dettagliato è un'applicazione comune delle finestre.

Infine, la possibilità di effettuare uno zoom (ingrandimento dell'immagine) e pan (muovere la finestra per selezionare una veduta da un display più grande) permette all'utente di avere una flessibilità maggiore in ambienti dove si faccia grande uso della grafica. Fino a questo punto la discussione è stata limitata ad oggetti bidimensionali. L'estensione ad oggetti tridimensionali richiede la capacità di gestire un maggior numero di cifre ed è necessario un moltiplicatore di tipo hardware. L'Am29116 combinato con l'Am 29516/517 (moltiplicatore 16 x 16) permetterà l'esecuzione di alcune operazioni grafiche tridimensionali. Gli algoritmi di rotazione tridimensionali contengono molte moltiplicazioni ed il tempo di moltiplicazione di 65 nsec dell'Am 29516/517 è abbastanza breve per effettuare i calcoli necessari in tempo reale. Le proiezioni bidimensionali di oggetti tridimensionali sono molto utili nelle applicazioni di Computer Aided Machining. Anche questo tipo di operazione contiene molte moltiplicazioni ed è necessario disporre di una capacità di moltiplicazione hardware in ambienti interattivi. ■

DESIGN ENTRY

Monolithic color palette fills in the picture for high-speed graphics

The first color palette chip puts a staggering array of hues at a designer's fingertips almost instantly. To boot, it boasts its own video output.

The graphics color palette is one of several modular blocks for building flexible, high-resolution graphics systems (see the Design Entry, p. 203). Standing alone, it eases programming, overlays text on graphics, and coordinates video-sync and blanking pulses.

A picture today is worth more than the customary 1000 words. On a color graphics system, it could—in theory—easily need over 1 million 24-bit words. That is certainly the case when such a system resolves an image into 1024-by-1280 picture elements and assigns every one 256 gradations each of red, blue, and green, making for a total of 16.8 million possible hues.

The color palette approach cuts this overwhelming memory requirement down to a manageable size by making only a small subset of the full spectrum available at any one time. Now, a complete subsystem implementing this technique for one color gun has been integrated onto a single chip.

Optimized for high-resolution color graphics systems, the Am8151 is essentially a 256-by-8-bit RAM plus an 8-bit digital-to-analog converter that is compatible with an RS-343-A video output. At any moment, three such chips yield no more than 256 out of 256^3 possible colors. But

Steven B. Sidman and **John C. Kuklewicz**
Advanced Micro Devices Inc.

Steven B. Sidman is a manager in AMD's analog and interface product-planning department in Sunnyvale, Calif. Earlier, he designed high-speed test systems at GenRad/STI, and before that he worked as a staff scientist at Lawrence Berkeley Laboratory. He holds a BSEE from Tufts University.

John C. Kuklewicz, who works at Kaiser Electronics in San Jose, Calif., is a former member of AMD's technical staff in analog and interface product planning. He earned a BSEE from Worcester Polytechnic Institute.

the host processor may rearrange the contents of each RAM at any time to create a different set of 256 colors (see "Portrait of a Color Palette," opposite). Each palette chip can be completely reloaded within 52 μs (it needs 200 ns to load a byte), leaving ample processor time for handling an array of 1024 by 1280 picture elements.

For the host processor, the palette approach slashes software overhead; for the system designer, it makes writing software easier. From the point of view of the application programmer, it makes colors much easier to manipulate. Moreover, the chip can be used independently or as part of a four-member family of bit-mapped graphics system ICs.

In good company

Although similar color encoders exist as hybrid circuits, the palette chip is the first IC to combine a 256-byte color map, a low-glitch d-a converter, three pixel-data and two monitor-control pipeline registers, a voltage reference, and all monitor control signals on a single chip (Fig. 1). In fact, it stands in for two 256-by-4-bit ECL RAMs, four 6-bit ECL registers, an 8-bit d-a converter and voltage reference, and various logic elements. That is just part of the picture; it also replaces 32 termination resistors (in single in-line packages) and a monitor-control IC.

More than simply cutting parts count, the chip also holds down the number of connections that must be made. These can become troublesome at high frequencies. Further, the IC's internal pipeline registers serve as buffers along its internal data paths. That arrangement relaxes the timing requirements for incoming signals and eliminates the need for external delay elements. The chip runs on a 103-MHz clock and delivers color to 60-Hz, noninterlaced displays of 1024-by-1280-pixel arrays. Separate controls allow text to overlay any graphics.

Timely interfacing

Internally, the chip's pipelined architecture simplifies the task of linking to its primary input, the color-map RAM, and its output, the d-a converter. A map-address register, for instance, holds pixel data—actually an address that is sent from the frame buffer. By storing the address, the register leaves most of the system clock cycle for generating the next address. Similarly, a color-data register holds the map's output and eases speed constraints on both the memory access time and the d-a converter's decoding circuitry. The d-a decoding register stores the converter's decoded input and deskews it, consequently helping reduce output glitches. Lastly, all monitor-control signals—High Sync (HSYNC), Vertical Sync (VSYNC), $\overline{\text{Blank}}$, Overlay (OVLY), and White or Black (W/$\overline{\text{B}}$)—are also loaded into registers to keep them synchronized with the Map Address inputs. Thus no external delay elements are needed for that purpose.

The fact that video-frequency inputs can handle either ECL or TTL also helps the input interface. One signal, LC (Level Change) actually controls the input threshold: when it is at +5 V dc, the inputs accept TTL; when it is at ground, the inputs work with 10K ECL. When set for the second, the inputs lend themselves to wired-OR multiplexing, which simplifies the interface over which the host processor changes the chip's color map. The signals that can switch between TTL and ECL thresholds are Map Address, OVLY, W/$\overline{\text{B}}$, CLK and $\overline{\text{CLK}}$.

Framing the picture

In normal operation, the palette chip either refreshes or updates the display with data (actually, indirect addresses) from the frame buffer. The buffer's parallel data is serialized before it is fed to the chip from a bank of video shift registers. The information enters through the Map Address inputs PD_0-PD_7 and accesses the 256-byte color map. Subsequently, the map's contents are applied to the d-a converter, which in turn produces an output for driving a CRT monitor.

Before generating the colors indicated by a bit-mapped frame buffer, the chip has to be loaded with a color map. To do so, the host processor must address the map, write new data, and possibly verify the entry.

The host loads and reads the color map through the graphics chip's bidirectional Color Data pins, which are controlled by the I/$\overline{\text{O}}$ and Enable ($\overline{\text{EN}}$) inputs. When $\overline{\text{EN}}$ is high, the data path is disabled and the pins remain in a high-impedance state. That makes it possible to con-

Portrait of a color palette

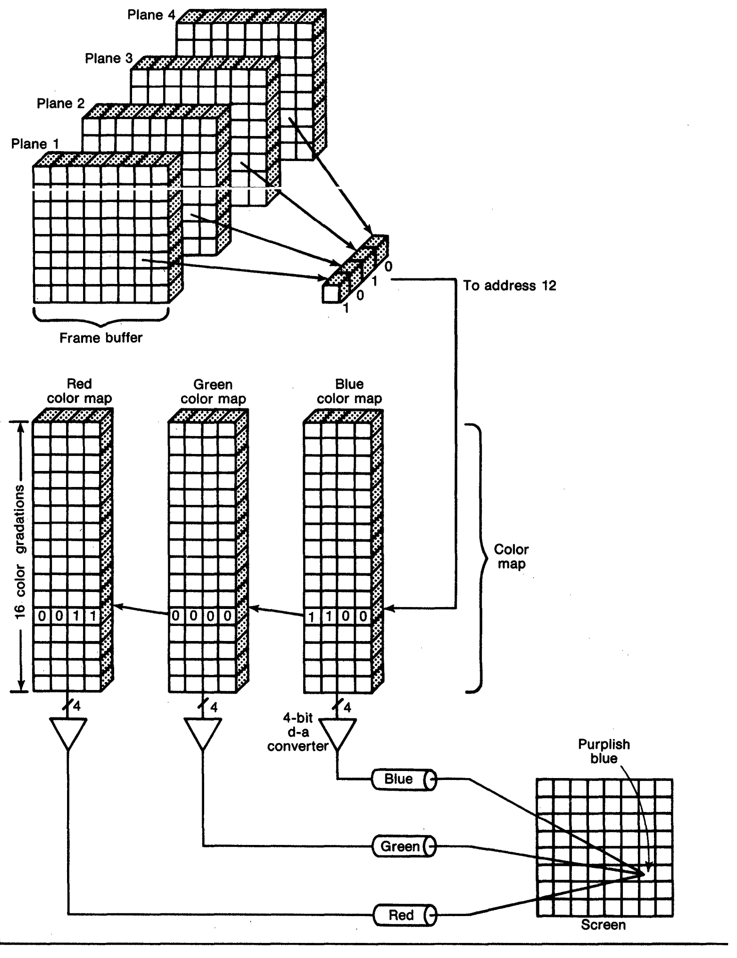
Digital color palettes, sometimes called palette mixers or encoders, simplify the problems involved in managing the massive number of color combinations and the associated memory space found in high-performance bit-mapped graphics systems. Typical examples are engineering workstations and image-processing equipment.

A raster-scanned display composes images out of picture elements of different relative intensities, or brightnesses. A digital word N bits long defines the brightness of each pixel, and a frame buffer N memory planes deep stores each word in accordance with each pixel's position in the display. (Every memory plane mirrors the organization of the pixel array.)

For a color system, each pixel requires at least three N-bit words, since three color guns contribute to each pixel's overall hue and intensity. Given an array of 1024 by 1024 pixels, each described by, say, three 8-bit words, the requisite 24 million bits of memory would be impossible for the host processor to manage.

A color palette—actually a look-up table stored in RAM—both eases the host's task and cuts the amount of memory needed in the frame buffer. At any one time, the RAM is capable of mapping all the gradations possible for each of the three primary colors. Consequently, instead of storing these color intensities directly, the frame buffer is free to store merely their locations within the color map. Each address points to the set of three red, blue, and green intensities that describe the color of a given pixel (see the figure). Thus color palettes are a form of indirect addressing.

A major advantage of this approach is that it alleviates the chore of changing colors because the system's host processor must write each color change to only one address in the palette. Its chief limitation is that it restricts the number of colors available to the display at any one time to a single set of red, blue, and green combinations—in other words, to the cube root of the entire spectrum of possible combinations. But the system's host processor can change the assortment of color intensities in the map, making the entire spectrum available at different times.



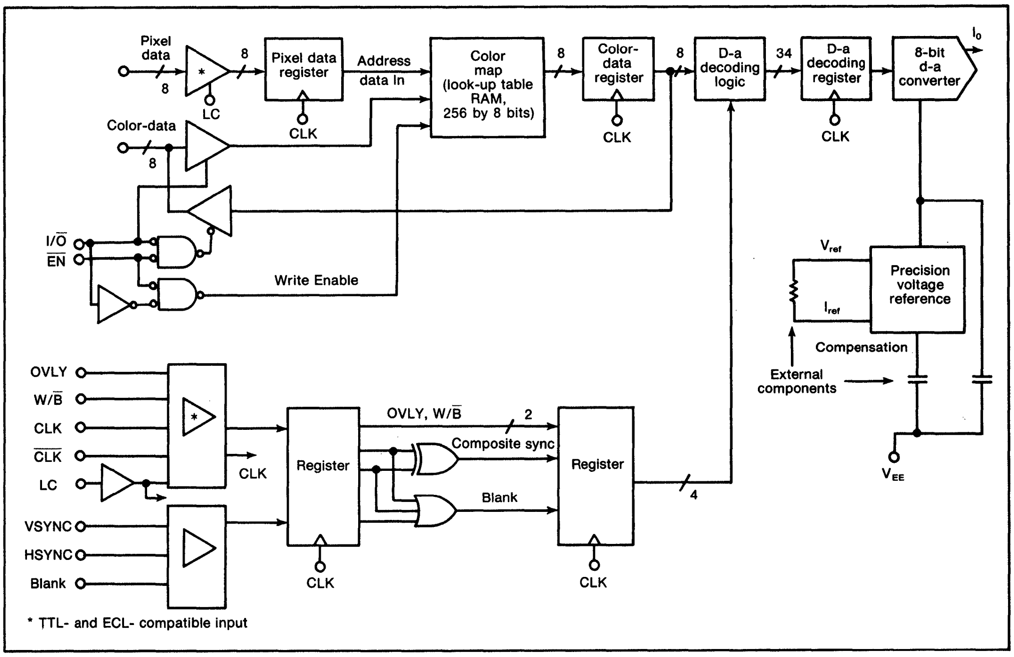
nect the Color Data pins directly to the host's data bus.

The processor selects bytes contained within the color map, singly or sequentially, by placing addresses on the chip's Map Address inputs (also referred to as Pixel Data inputs) and asserting the I/O and EN signals. Because the address inputs are also driven by the frame buffer (through the video shift registers), they must be multiplexed between the buffer and the processor to refresh and update the display. A bank of eight 2-to-1 TTL-level multiplexers is sufficient for that chore.

No multiplexers need apply

When ECL drives the Map Address inputs, however, multiplexers may not be needed. The ECL outputs of the video shift registers and the host processor can be tied together directly (through TTL-to-ECL translators), forming a wired-OR connection. Then, when the processor accesses the color map, it is only necessary that the outputs of the video shift registers be at a logic low. Conversely, when the frame buffer is addressing the map, the processor's translators must stay low. No matter how the processor enters the color map or for what purpose, the selected locations will be displayed unless the chip's Sync or Blank inputs are asserted.

The chip's HYSNC AND VSYNC inputs cause the converter's output to assume the proper voltage levels for synchronizing or blanking the display's sweep (Fig. 2). The two inputs are exclusive-ORed within the chip. In consequence, when either goes high, the converter's output is unconditionally driven to the composite sync level. But when both inputs are asserted, the horizontal sync pulses generate pulses that alternate between sync and blank



1. The Am8151 color palette is the first IC to combine a 256-byte color map, a low-glitch digital-to-analog converter, five pipeline registers, a voltage reference, and all functions for controlling a color CRT monitor. The 40-pin DIP replaces seven ICs and four single in-line packages of pull-up resistors. It can be used independently or as part of a four-member set to establish a high-resolution bit-mapped graphics system.

levels during the vertical retrace period, keeping the horizontal oscillator locked into its proper frequency. Asserting the chip's Blank input forces the converter's output down to the blank signal line level. That occurs regardless of all other inputs, except when HYSNC and VSYNC are driving the output to the still lower sync level.

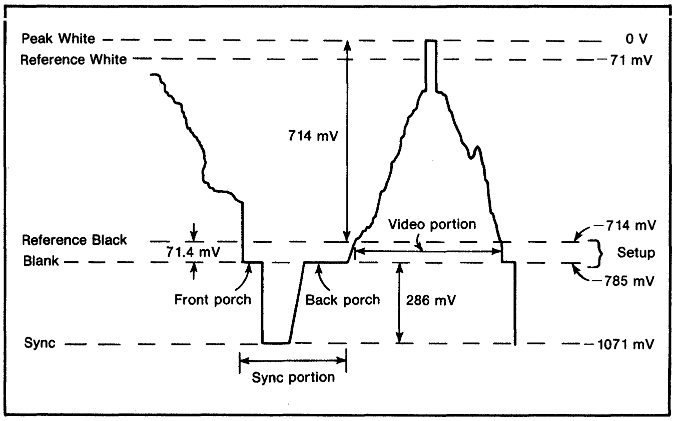
Whiter than white

The OVLY and W/$\overline{B}$ inputs overlay text onto graphics. When both are high, the graphics chip's output is driven to peak white, a condition that is 10% whiter than the reference value and allows white text to be written over white graphics. When three palette chips are configured in a RGB system, it is possible to write text over graphics in eight colors by supply independent W/$\overline{B}$ lines for each chip (Fig. 3).

Invoking an overlay allows other video sources to be multiplexed onto the display because the chip's output in the peak white condition is 0 V dc, which corresponds to a high-impedance state. As a result, when the palette is operating in this condition, another source can drive the monitor with no interference.

When changing colors, it is possible to load a single word into the color map in about 200 ns. It can be read back in 110. Thus all three chips in a color system can be loaded in less than 160 μs and read back in 90. If the host processor can carry out the necessary 768 writes and an equal number of reads rapidly enough, the entire color map can be altered and verified during the vertical retrace time—approximately 1.5 ms.

Transfers that involve the color map must take into account the chip's pipeline registers. Reading from or writing to the map does not instantly change the displayed image because the pipelines delay the processing of individual



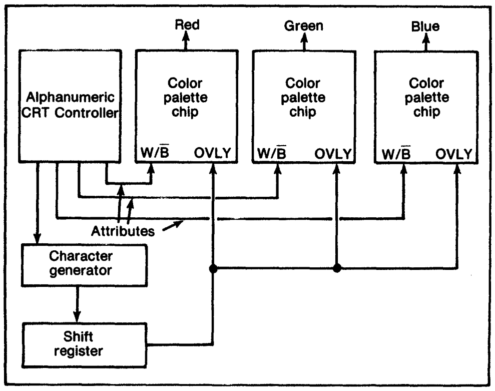
2. The palette chip conforms to video standard RS-343-C and even has a peak white level for superimposing white text on white graphics. it contains a nearly glitch-free d-a converter that produces the video output waveform by sourcing current into a reference resistance. The resistance also sets the voltage for black and white reference levels, blanking, and setup. (The last is the difference between the blank and black reference levels).

colors by three clock cycles. For the same reason, it takes two clock cycles to load an initial byte into the color map. Once the pipeline is filled, through, successive changes occur like DMA transfers: one byte to a clock cycle.

An 8-bit d-a converter links the palette to a video monitor. The latter is designed to minimize output glitches and eliminates the need for deglitching circuitry. Glitch energy is kept to 50 pV-s or less by segmenting the converter into two separate, but matched, d-a converters. One accepts the four most significant input bits; the other receives the four least significant bits. The full-scale output of the second d-a converter is 1/16 of the full-scale output of the first d-a converter. Importantly, each converter is decoded, which means that instead of the four binary-weighted current sources that are found in most d-a designs, each of the palette's two converters contains 15 equal sources—one for each analog output increment.

The converters feature such low glitch energy because internal current changes are limited to either opening or closing current sources, and exclude the simultaneous occurrence of both. In contrast, binary-weighted converters attempt to switch on some currents as others are turned off. When skewed, these opposite current switches can cause a converter's worst glitches. What's more, the weighted currents create glitches in proportion to their weight, up to 1/2 MSB. The two decoded converters, however, never switch a current larger than 1/16 of the full-scale output.

The converter's output also settles quickly, within 12 ns, contributing to the graphics chip's high speed. Its dual-decoded architecture, bipolar process, and careful layout not only help it settle quickly but also further limit glitch energy. Also, for short paths the converter's current output can directly drive low-impedance loads (50- or 75-Ω), eliminating the need for a buffer.



3. The OVLY and W/B̄ inputs are the keys to the palette's ability to superimpose text on graphics. When OVLY is asserted, a high on the W/B̄ input generates a 10% "overwhite" that writes white text against a white background. In a three-gun color system, keeping the three W/B̄ lines independent furnishes a total of eight possible colors for the superimposed text.

On-board reference

A precision voltage reference is carried by the palette chip, as part of the d-a converter. The reference sets the converter's full-scale video output so that it is compatible with the display monitor. Placing a resistor between the Reference Out and Reference In pins (REFOUT and REFIN, respectively) selects the video output voltage. For instance, a 2-kΩ external reference resistance resistor sets the converter's full-scale output to the RS-343-C composite sync level, −1.071 V dc (into a 75 Ω load).

For other video standards, the requisite value of a resistor is found with the equation $I_{FS} = 28.56\text{ V dc}/R_{ref}$, where I_{FS} is the full-scale current in milliamperes and R_{ref} is the reference resistance in kilohms.

The relationship of intermediate video currents to I_{FS} and R_{ref} hinges on two facts: I_{FS} corresponds to the composite sync level, and the chip resolves the video signal between reference white and reference black into 256 equal steps. For that reason, other important currents are:

- reference white, $I_{REFWHT} = 1.7556\text{ V dc}/R_{ref}$
- reference black, $I_{REFBLK} = 17.676\text{ V dc}/R_{ref}$
- blanking, $I_{blank} = 19.421\text{ V dc}/R_{ref}$
- setup, $I_{setup} = 1.7556\text{ V dc}/R_{ref}$

The last current, I_{setup}, is the difference between the blank and the reference black levels.

The precision voltage source's Reference In

Cover: Color palette chip

and Reference Out pins also enable several palette chips to be linked to the same reference. In doing so, a designer can better match the color intensities among all of the chips. Specifically, the REFOUT of one palette chip may be used to drive the REFIN of two other chips. Because the absolute gain of each palette's d-a converter is within 1% of its setting, the full-scale outputs of any three converters driven from the same reference will match each other to within 2% (excluding mismatches caused by other current-setting or load resistors).

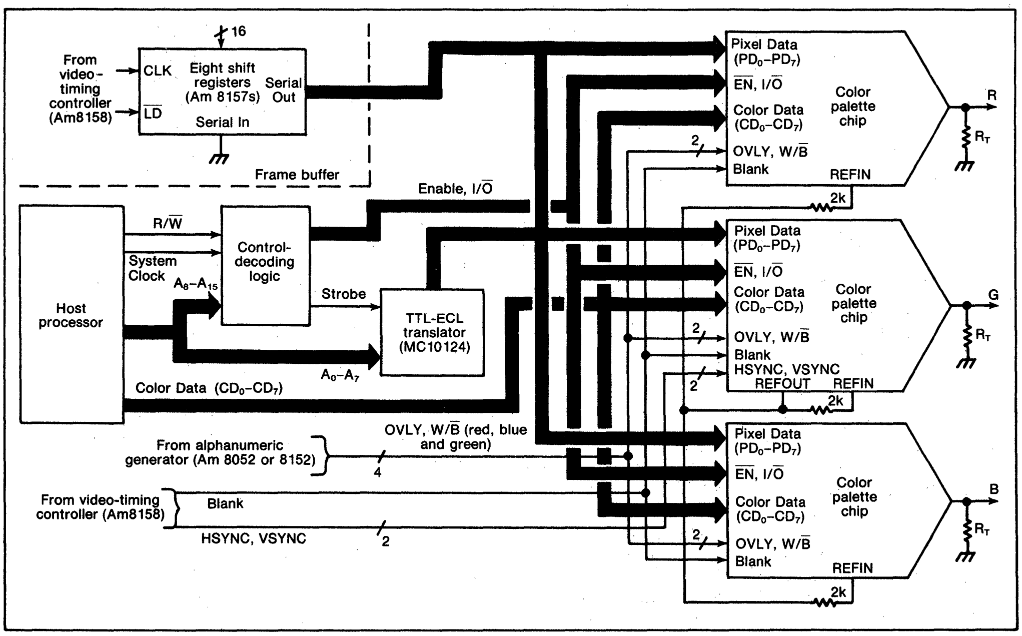
Bit mapped and blocked out

A simplified block diagram of a bit-mapped graphics system is an easy way to show the signal paths leading to the palette chip (Fig. 4). In the circuit, a wired-OR handles the multiplexing that lets the host processor (through the aforementioned TTL-ECL translators) and the frame buffer (through eight video shift registers) share access to each palette's Memory Address inputs.

Since the video shift registers work in conjunction with the other members of the graphics chip family, the register's outputs can be made low whenever a video timing controller asserts the Blank signal. That sequence of operations prevents the shift registers from being updated during the retrace intervals. Consequently, the host processor can address the color map during these intervals, allowing the user to change or check the available colors without the monitor showing any disturbance.

Part of the overall system, a control-decoding block, generates a common strobe for all TTL-ECL translators from the processor's Read and Write lines, system clock, and high-order address bits. Here, the translators are set up so that a low on the strobe line forces the ECL outputs low.

The decoding block also drives the $\overline{EN}$ and I/$\overline{O}$ palette-state control signals. The bidirectional Color Data Lines from all three palettes



4. In a simplified block diagram of a color bit-mapped graphics system, a host processor (through TTL-ECL translators) and a frame buffer (through video shift registers) are wire-ORed to three palette chips. To avoid disrupting the display, the chip's color maps can be loaded or changed while the color gun is vertically retracing its beam. One palette drives each gun.

are connected directly to the host processor's data bus.

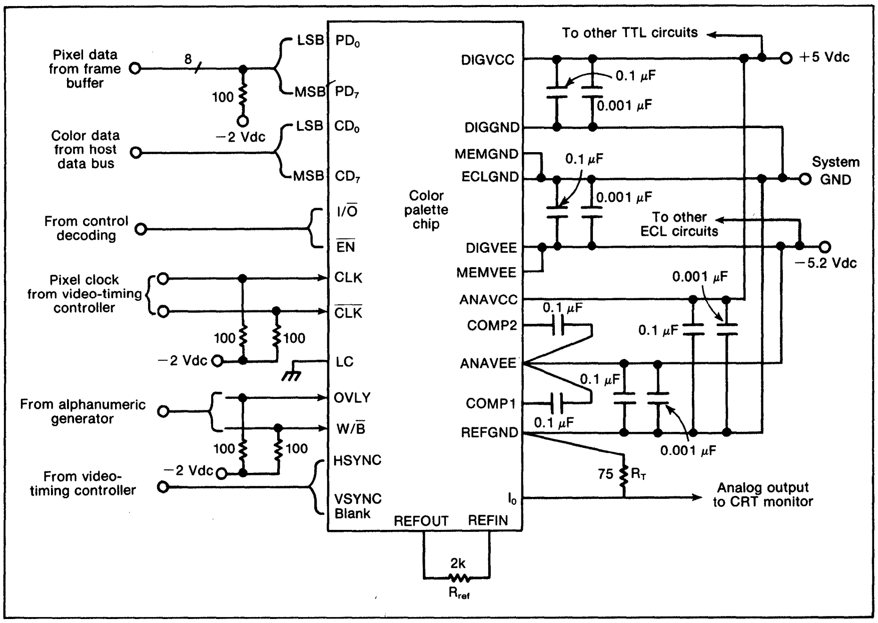
A video-timing controller generates the Blank, VSYNC, and HSYNC monitor-control signals and transmits them to the monitor through the palette. In most monitors, composite sync information is decoded from only the green channel because just one set of sync pulses is needed. Blanking information, on the other hand, goes to all three color channels. An alphanumeric CRT controller subsystem supplies the OVLY and W/$\overline{B}$ inputs to the palette.

Because of the palette's high speed, care must be exercised when linking it to surrounding circuitry. To this end, the chip has several different ground and power connections to help limit the effects of noise. There are four ground pins: one each for TTL digital (DIGGND), ECL (ECLGND), memory (MEMGND), and reference (REFGND) circuits. The first should be tied to the ground point used by all TTL circuits on the board. ECLGND and MEMGND should be connected at the chip and then to the board's ECL ground point. And REFGND should be joined to the ground system as close as possible to the analog V_{EE} supply point, ANAVEE, to minimize noise from digital ground currents. A good point of connection is where the ground systems are brought onto the board.

Power points

A typical pin-by-pin hookup for ECL-compatible inputs (Fig. 5) is quite similar to that used for TTL-compatible inputs. With the latter, the LC input would be tied to +5 V dc at the digital V_{CC} pin, DIGVCC, instead of to ground. Further, the 100-Ω resistors terminating at −2 V dc would be eliminated, and the CLK input would be tied to the digital ground point.

To minimize noise, DIGVCC is linked to 5 V dc and can share a common connection with other TTL circuits on the same board. But the analog V_{CC} input, ANAVCC, should be con-



5. The palette's 103-MHz speed demands that a designer pay special attention to minimizing noise, especially from ground loops. The power pins for TTL (DIGVCC), memory (MEMVEE), and analog (ANAVEE and ANAVCC) supplies are at the right. Individual pins are used for TTL, ECL, memory, and voltage-reference resistance.

nected as closely as possible to the 5-V dc digital supply. In addition, the digital V_{EE} and memory V_{EE} inputs (DIGVEE and MEMVEE, respectively) are best tied in to −5.2 V dc and connected in common with other ECL circuits on the board. As with ANAVCC, however, the analog V_{EE} connection is best made as near as possible to its respective −5.2-V dc source.

All power supply lines, whether analog or digital, should have not only a low dc resistance but also a low impedance, even at frequencies in the hundreds of megahertz. This requirement means placing high- and low-frequency bypass capacitors (0.1 µF and 0.001 µF, respectively) as closely as is practicable to the chip's power supply and ground pins.

Low-impedance grounds are equally important, and a ground plane for ECL is recommended. In addition,the reference ground system should have wide, thick conductors to introduce as little inductance as possible into the system. To avoid signal-induced ground currents, a 75-Ω resistor to REFGND should terminate output I_0 because its complement, $\overline{I}_0$, returns to REFGND within the chip.

Similarly, all signal paths and components should be laid out paying careful attention to good high-frequency practices. Short lines built as microstrips work best for all digital inputs like Map Address, OVLY, $W/\overline{B}$, CLK, and $\overline{CLK}$. Moreover, termination resistors that match each line's characteristic impedance should be tied in as near as they can to the chip's pins.

Extra care is also called for at REFIN and compensation points COMP1 and COMP2, because their high impedance makes them particularly susceptible to externally induced noise. Links to these points must be as short as possible, even at the expense of lengthening connections on the other side of the chip. Capacitors at COMP1 and COMP2 should be located particularly close to their respective pins, and their dc ends should be terminated at ANAVEE in a single-point, or star, fashion to avoid ground current loops.□

Three chips, plus palette, aid graphics designers to paint detailed pictures

Three coordinated building-block ICs, organized around a bit map, ease the job of designing graphics systems that can resolve upward of 1024 by 1024 pixels.

This article, which is related to the cover story on a color palette chip (p. 187), deals with the three other chips in a family of high-performance graphics building blocks. They are powerful and programmable; thus they do not tie a designer to a particular graphics processor, image updating scheme, or video standard.

Like the harried air-traffic controllers who allocate airspace, graphics system designers must allocate access to the address space in a display's bit-mapped memory. In both cases, timing is everything, and often the designer's job can seem as much of a juggling act as the air-traffic controller's. To satisfy a graphics system's conflicting memory access demands, designers can now call on the Am8150 display refresh controller for assistance.

Andrew Daniel, Adrian Sfarti, and **Achim Strupat**
Advanced Micro Devices Inc.

Andrew Daniel is a product planner and an applications engineer with AMD's advanced graphics group in Sunnyvale, Calif. He has been with the company since 1982 and now designs and develops VLSI graphics chips. He holds a BSEE and an MSEE from Rensselaer Polytechnic Institute.

Adrian Sfarti is a section head of the advanced graphics group for product planning. He holds a BSEE and an MSEE from the Polytechnical Institute of Bucharest in Romania and a BS in mathematics from the University of Bucharest.

Achim Strupat has been a product planner and an applications engineer at AMD since 1983. He is responsible for specifying high-performance VLSI graphics components and holds an MS in computer engineering from the RWTH Aachen in West Germany.

It is the most complex of three chips designed to support the Am8151 color palette, an IC that conserves memory space. Three color palettes together need only 8 memory planes to display 256 hues simultaneously out of 16.8 million. The two other supporting chips are the Am8157 video shift register, which serializes parallel data from the bit map and sends it to a color palette, and the Am8158 video timing controller, which synchronizes the system and a sends timing signals to a display monitor. Together, the four ICs ease the job of designing high-resolution graphics systems—those with at least 1024 by 1024 pixels.

High on the refresh controller's list of strengths is its ability to execute any of three different memory arbitration schemes. Perhaps the most interesting is the one that interleaves memory refreshing and display updating cycles, a first for graphics chips. This mode ensures flicker-free screen refreshing while maintaining a high bandwidth to update the video image.

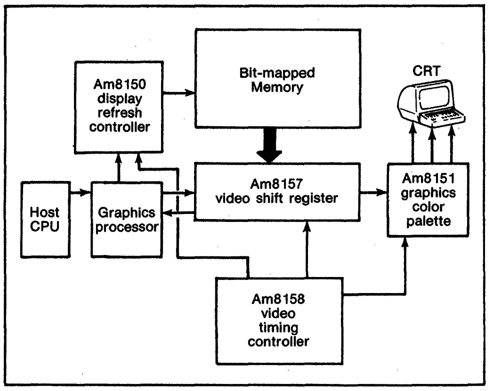
Rally round the bit map

The bit map is the focal point of a graphics system built around the Am815x family (Fig. 1). Working together, the chips can be programmed by the system CPU to adjust the display memory's size, the memory chip's organization, and the video frame's size and position. For example, the number of memory planes, bits per pixel, and bits per memory word all can be expanded in increments.

The display refresh controller addresses the bit map to refresh both the CRT screen and the dynamic RAMs that make up the bit map. In addition, the controller translates linear updating and refreshing addresses into multiplexed row and column addresses.

These processes all use the same address path to the display memory. Thus the time it takes to cycle through a single access, as well as to arbitrate among the three processes, is critical. Excessive flickering, sluggish image motion, or no image at all may result if the bit memory is not accessed when needed.

The problem of meeting each demand for access on time is complicated by the different address sources and data-path requirements of refreshing and updating. On the one hand, the addresses for refreshing the video display and


Am8150 display refresh controller
Bit-mapped Memory
CRT
Host CPU
Graphics processor
Am8157 video shift register
Am8151 graphics color palette
Am8158 video timing controller

1. A bit-mapped memory is the focal point of a high-resolution graphics system built with the Am815x family of chips. The display refresh controller, which juggles memory accesses, and the video shift register, which serializes parallel data from the memory, connect directly to the bit map. The video timing controller synchronizes the digital and video signals, and the graphics color palette makes a wide choice of colors available with relatively little memory.

dynamic RAMs come from the refresh controller, and the complete set of display memory words are accessed in predictable cycles. On the other hand, the addresses that update the bit map originate at the system's graphics processor. They are bit-oriented and random, and they involve updating a video image by, for example, drawing vectors and circles or moving blocks of pixels.

Addressing ability

Because of the inherent differences between refreshing and updating an image, the display refresh controller lets the designer decide how best to accommodate each. It offers three programmable arbitration modes: interleaved updating, retrace updating, and update overriding.

In interleaved updating, about 50% to 70% of the frame time is available for updating the video. This mode supplies a flicker-free video screen and simultaneous high bandwidth for updating. To achieve this mix, video refreshing and updating cycles alternate during the active video time (the interval when the CRT's display is scanned). The retrace time is shared between updating the display and refreshing the dynamic RAMs.

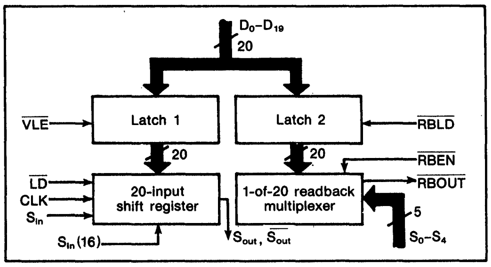
In retrace updating, about 10% to 20% of the time is spent updating the display. Here the active video time is devoted exclusively to refreshing the video. Updating shares the retrace time with the dynamic RAM refreshing cycles.

In contrast, update overriding, as its name implies, gives top priority to updating the display: about 95% of the time is available for that. On the other hand, video refreshing—and therefore any visible image—is sacrificed to the need for high bandwidth to change the bit map. When updating is complete one of the other two modes must be activated for the displayed image to return.

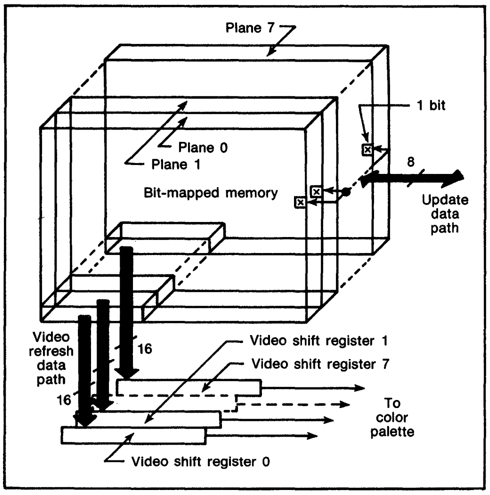
In all modes, the dynamic memories are refreshed during the horizontal retrace time. A user-programmable number of dynamic memory refresh cycles are used for each horizontal sync pulse.

Whereas the video-refresh and memory-refresh addresses are generated by the controller, the image-updating addresses enter the controller from the graphics processor; thus a

Graphics chip set



3. The video shift register has two main functional blocks: a 20-bit shift register that serializes bit-mapped data and sends it to the monitor (or optionally to a palette) and a readback multiplexer that sends pixel information to the host processor. Each block has a latch to ease timing restrictions.



4. An eight-plane display memory with a video shift register at each plane satisfies conflicting memory access requirements. The eight 16-bit parallel data paths help meet the demands for video refreshing of 125 MHz per plane. A narrower, 8-bit path allows bit-by-bit access of each plane for updating individual pixels.

read very quickly. For example, about 125 million pixels a second must be read to refresh a noninterlaced display that generates 1280 by 1024 pixels (horizontal by vertical) at a 60-Hz rate. This leaves only about 8 ns to access each pixel.

Because conventional dynamic RAMs have access times of about 100 ns (in page mode), data must be read several bits at a time to meet the 8-ns/pixel limit. Then the parallel data must be serialized to drive a monitor. All of these requirements are met by the video shift register.

The register is well equipped to accept parallel data and put it in serial form. It also offers a way to read back a set of retrieved pixels to the graphics processor. Reading data back can occur simultaneously with—and independently of—the serialization and transmission of data. To keep serialization independent of reading back, the video shift register is divided into two main blocks, one for each major function (Fig. 3).

Taking 20-bit chunks

In a system, one video shift register is dedicated to each plane of the display memory (Fig. 4). Each shift register, in turn, accepts up to 20 consecutive bits at a time (although 16-bit words are a particularly convenient width).

Moreover, word widths of more than 20 bits can be attained if multiple shift registers are cascaded. To add a shift register, the Serial Out (S_{out}) port of one chip is connected to the Serial In (S_{in}) port of the next. In this way each cascaded shift register loads and serializes an additional 16 or 20 consecutive bits from a memory plane.

Regardless of their width, parallel words are loaded from the display memory bus into the shift register when $\overline{LD}$ is asserted. Words also may be preloaded into a built-in latch to relax the incoming data's setup and hold requirements. The latch is controlled by the Video Latch Enable input ($\overline{VLE}$). Whichever way the data gets into the shift register, it is sent out serially and synchronously with the chip's Dot Clock input (CLK).

Dedicating at least one shift register to each plane nas two major advantages. First, adding memory planes is relatively easy, necessitating three-input multiplexer is needed in the refresh controller (Fig. 2). The multiplexer selects one address and sends it to the display memory as a row-address and column-address pair. The controller also issues correctly timed Row Address Strobe ($\overline{\text{RAS}}$) and Column Address Strobe ($\overline{\text{CAS}}$) signals. In all, it has four $\overline{\text{RAS}}$ signals with which to expand the memory plane up to four banks.

Expanding, panning and scrolling

The refresh controller also allows for smooth panning and scrolling, which in turn permit the screen to scan across a stored image that is larger than the screen itself. These effects are possible because the refresh controller can generate video refresh addresses for a small screen within a large display memory.

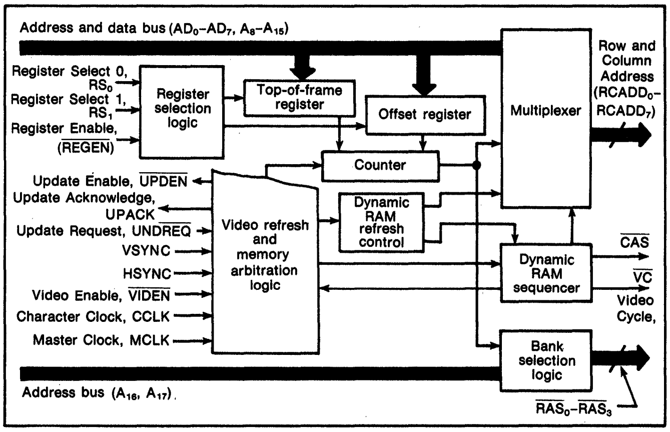
Two user-programmable registers define the display memory's screen size and position within a larger memory. A top-of-frame register stores the screen's starting location in memory. Subsequent addresses within a row are generated by incrementing the top-of-frame value. An offset register stores the number of locations between the last word of one row of the screen and the first word of the next row. The first address of the second row is the sum of the offset register value and the last address of the first row.

To achieve panning or scrolling, the contents of the top-of-frame register is incremented (that is, the register is reloaded) before every frame. For either effect, each new entry starts the screen at a different position.

Speedy and serial

As video refreshing sequentially addresses all graphical data in the memory, the video shift register receives the accessed data and sends it to the screen. This occurs during the active video portion of the sweep.

To refresh the video image in a high-resolution system, the display memory data must be



Address and data bus (AD_0–AD_7, A_8–A_{15})
Register Select 0, RS_0
Register Select 1, RS_1
Register Enable, ($\overline{\text{REGEN}}$)
Register selection logic
Top-of-frame register
Offset register
Counter
Multiplexer
Row and Column Address ($RCADD_0$–$RCADD_7$)
Update Enable, $\overline{\text{UPDEN}}$
Update Acknowledge, UPACK
Update Request, $\overline{\text{UNDREQ}}$
VSYNC
HSYNC
Video Enable, $\overline{\text{VIDEN}}$
Character Clock, CCLK
Master Clock, MCLK
Video refresh and memory arbitration logic
Dynamic RAM refresh control
Dynamic RAM sequencer
$\overline{\text{CAS}}$
$\overline{\text{VC}}$
Video Cycle,
Bank selection logic
Address bus (A_{16}, A_{17}).
$\overline{\text{RAS}_0}$–$\overline{\text{RAS}_3}$

2. The display refresh controller orchestrates accesses to the bit-mapped memory's addresses. Demands for access come from the video refresh section, which sustains the displayed image; the dynamic RAM refresh circuitry; and an external graphics processor, which updates the bit map. The controller is the first graphics chip to interleave video refreshing and bit-map updating during an active video display.

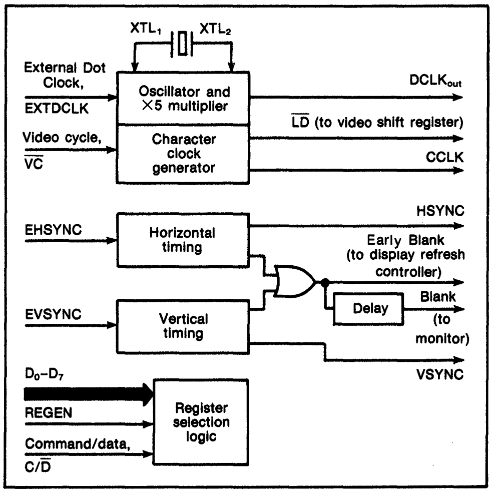
little more than additional memory chips and shift registers. Second, the system's bandwidth requirement does not increase with additional planes.

One bit at a time

In a multiplane system, each shift register sends out one bit of a pixel at a time. Therefore, at any given moment the sum of the bits out of all the shift registers defines the pixel in question. From the shift register's serial output port, the bits go either to the color palette chip or to the monitor.

The second major function of the video shift register is to feed bits back to the host processor. Invoking the shift register's readback feature begins by asserting the Readback Load input (RBLD), which stores one display memory word in the readback latch.

Up to 20 pixels are available; selection lines S_0 through S_4 choose which of the 20 will be read out of each latch. Provided the value of the selection lines is less than 20 (values of 20 or more are ignored), asserting the Readback Enable signal ($\overline{\text{RBEN}}$) places the selected bit on the chip's Readback Out line (RBOUT). Therefore, when the RBOUT lines of each shift register—and therefore each memory plane—are connected to a different line of a system bus and $\overline{\text{RBEN}}$ is asserted, the entire pixel is presented to the graphics processor.


XTL$_1$
XTL$_2$
External Dot Clock, EXTDCLK
Oscillator and ×5 multiplier
DCLK$_{out}$
$\overline{\text{LD}}$ (to video shift register)
Video cycle, $\overline{\text{VC}}$
Character clock generator
CCLK
HSYNC
EHSYNC
Horizontal timing
Early Blank (to display refresh controller)
Blank (to monitor)
Delay
EVSYNC
Vertical timing
VSYNC
D_0-D_7
REGEN
Register selection logic
Command/data, C/$\overline{\text{D}}$

5. The fully programmable video timing controller generates horizontal and vertical synchronization pulses, blanking pulses, and dot and character clocks. The dot clock and synchronization pulses can be externally triggered to coordinate the action of other graphics circuits.

All timing under one roof

The third member of the family is the video timing controller. It issues all the timing signals needed by the other components of the bit-mapped system, including those for the video monitor (Fig. 5).

Ultimately, all timing signals are generated from the chip's on-board dot-clock oscillator. The clock, capable of reaching up to 125 MHz for high resolution, is crystal-controlled. Although an on-board oscillator simplifies the system, a designer can elect to supply a dot clock from one of two sources. The first is a TTL-compatible signal that, like the crystal, is one-fifth of the desired dot-clock frequency. The second is an ECL-compatible signal at the full dot-clock frequency.

A character clock output and an $\overline{\text{LD}}$ signal derived from the dot clock, supply the timing signals for the display refresh controller and video shift register, respectively. With these signals, the chips address and latch display memory words during the refresh cycles.

To adapt to different video standards and screen sizes, the timing controller's video signals have programmable parameters. For example, the lengths of the Horizontal and Vertical Sync pulses (HSYNC and VSYNC), as well as their front and back porch lengths, are programmable, as are the number of pixels to a scan line and scan lines to a frame. The chip also produces a video blanking signal for the monitor and a Video Enable (VIDEN) signal for the display refresh controller.

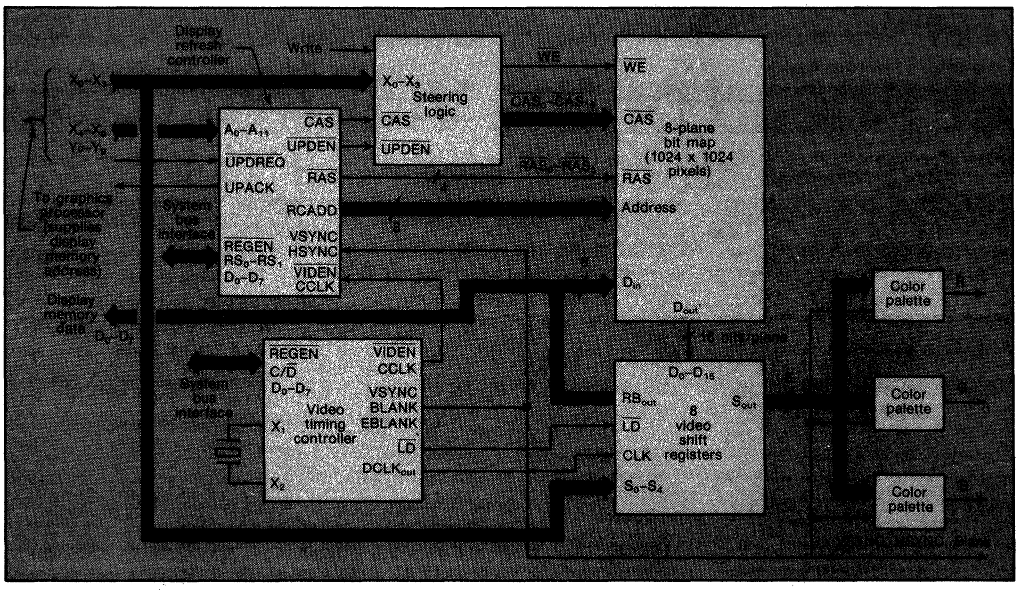
As an alternative, the video pulses from the timing controller can be synchronized to external sources. That permits the chip set to operate with other devices, as needed, to mix text and graphics, or to synchronize the display to the power-line frequency. For greater flexibility, the external sources, which are applied to inputs EHSYNC and EVSYNC, can be asynchronous to the character and dot clocks.

In a typical application, the four chips form the core of an efficient pixel-oriented graphics system that resolves the conflicting requirements of refreshing the video screen, refreshing the dynamic RAMs, and updating the display memory (Fig. 6). For low-overhead video refreshing, the system lays down a wide data path from the display memory to the CRT; for efficiently updating the bit map, it provides addresses to individual pixels.

The system has eight 1024-by-1024-bit memory planes, each made up of sixteen 64-kbit RAMs. The display refresh controller orchestrates the video refreshing, memory refreshing, and video updating. Eight video shift registers, one for each memory plane, serialize the data and are a means for reading back the contents of the memory. The shift registers' serial outputs drive three color palette chips, and the timing controller produces all the timing signals. A separate graphics engine generates updated addresses and manipulates the stored images; a sprinkling of additional components, embodied in the steering logic, completes the system.

The pause that refreshes

To refresh the screen at the standard rate of 60 frames/s, a complete frame must be sent to the monitor every 16.6 ms, requiring that a new line be scanned every 16 μs. With 1024 pixels in a line, the bit rate for refreshing the screen would seem to be 15 ns/bit, but after horizontal and vertical retrace times are subtracted, the actual bit rate is more like 10 ns/bit. Therefore, if the memory is read in the page mode and its read cycle time is 160 ns, then at least 16 bits must be read from the memory simultaneously to re-



6. Together, the four-chip set reduces the parts count of a 1024-by-1024 pixel graphics system. The added steering logic helps the display refresh controller by distributing the controller's CAS output among individual memory planes so that a graphics processor can update individual pixels. The system uses eight video shift registers, one for each memory plane, and three color palettes, one for each color gun.

Steering logic truth table

Input		Output		
$\overline{\text{UPDEN}}$	X_0 X_1 X_2 X_3	Active $\overline{\text{CAS}}$ lines	$\overline{\text{WE}}$	Process
1	X X X X	None	1	Dynamic RAM refreshing
1	X X X X	All	1	Video refreshing
0	0 0 0 0	$\overline{\text{CAS}}_0$	1,0	Video updating
↑	0 0 0 1	$\overline{\text{CAS}}_1$	↑	↑
	0 0 1 0	$\overline{\text{CAS}}_2$		
	0 0 1 1	$\overline{\text{CAS}}_3$	(1 for	
	0 1 0 0	$\overline{\text{CAS}}_4$	read cycle,	
	0 1 0 1	$\overline{\text{CAS}}_5$	0 for write	
	0 1 1 0	$\overline{\text{CAS}}_6$	cycle)	
	0 1 1 1	$\overline{\text{CAS}}_7$		
	1 0 0 0	$\overline{\text{CAS}}_8$		
	1 0 0 1	$\overline{\text{CAS}}_9$		
	.	.		
	.	.		
	.	.		
↓	1 1 1 1	$\overline{\text{CAS}}_{15}$	↓	↓

fresh the screen adequately.

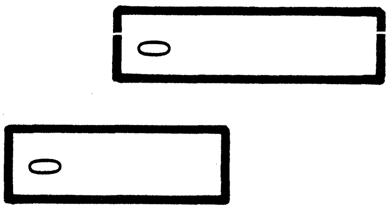
With that requirement, it is especially convenient to organize the planes around sixteen 64k-by-1 RAMs and to concurrently take one bit from each RAM to satisfy the video refresh requirements. However, this organization is not convenient for updating the memory, because updating demands access to individual pixels and thus to individual memory bits.

For video updating, therefore, the $\overline{\text{CAS}}$ signals are steered to each of the 16 chips individually. Addressing a single bit is then accomplished by time-multiplexing the display memory's X and Y coordinates to generate the row and column addresses for the 64-kbit RAMs. Specifically, coordinates X_0 through X_3 activate one of the 16 $\overline{\text{CAS}}$ lines, and X_4 through X_9 and Y_0 through Y_9 address an individual location within a selected RAM. A pixel is ready to be written when the $\overline{\text{RAS}}$ and Write Enable lines ($\overline{\text{WE}}$) for all 128 RAM chips are activated; then, with the help of the steering logic, only one of 16 $\overline{\text{CAS}}$ signals is asserted and a pixel is written (see the table, above).

To read back the memory contents, data can again be accessed in 16-bit words (representing 16 pixels) from each memory plane and latched into eight corresponding video shift registers. Select lines pick one bit of the 16 in each readback latch and assertion of the shift register's Readback Enable activates the path.□

TWO-CHIP SET TACKLES DISK CONTROL PROBLEMS

Handling data separation functions before passing data to the disk controller lets a two-chip disk controller supply high data transfer rates, while ensuring design flexibility.



by Mark S. Young, Pradeep Padukone, and Neil Adams

Designers attempting to integrate Winchester and floppy disk drives into systems are faced with a complicated and demanding task. To coordinate the activities of an asynchronous, analog device (the disk) with a synchronous, digital interface (the computer system), several major problems must be solved. Special analog circuitry is necessary to synchronize and decode data pulses coming from the disk during a read operation. Complex algorithms are required to improve relative read or write speeds. Attention must be paid to data recovery methods as well as to error detection and correction. In addition, software support and system interface overhead must be carefully considered in order to ensure effective communication with the computer system.

These problems have been addressed by specialized disk interface chips. A lack of sufficient onboard intelligence, however, limited their success while higher bit densities, advanced encoding schemes, and faster data rates made the job more difficult. In a renewed attack on the problem of providing a cost-effective interface between disk and computer system, a two-chip controller set segregates the data separation function from other tasks to maximize disk I/O.

Mark S. Young is a product planning engineer in charge of the hard disk controller chip at Advanced Micro Devices, Inc, 901 Thompson Pl, Sunnyvale, CA 94086. He holds a BA in computer science from the University of California at Berkeley.

Pradeep Padukone is senior product planning engineer at AMD. He holds a BE from Bangalore University and an ME in electrical communication from Indian Institute of Science, both located in Bangalore, India, and a PhD in systems engineering from Oakland University, Rochester, Michigan.

Neil Adams is a senior engineer at AMD. He holds a BS in physics and electronics from Manchester University, Manchester, England.

Addressing disk control issue

The Am9580/Am9581 chip set addresses issues related to controlling disk drives. The Am9580 hard disk controller (HDC) is a MOS VLSI Winchester floppy disk controller, and the Am9581 is a bipolar disk data separator (DDS). The two-chip set performs three basic functions: disk format/control, data separation/data encoding and decoding, and data transfer to and from system memory. When used together, the chips provide a complete solution to Winchester/floppy disk control (Fig 1).

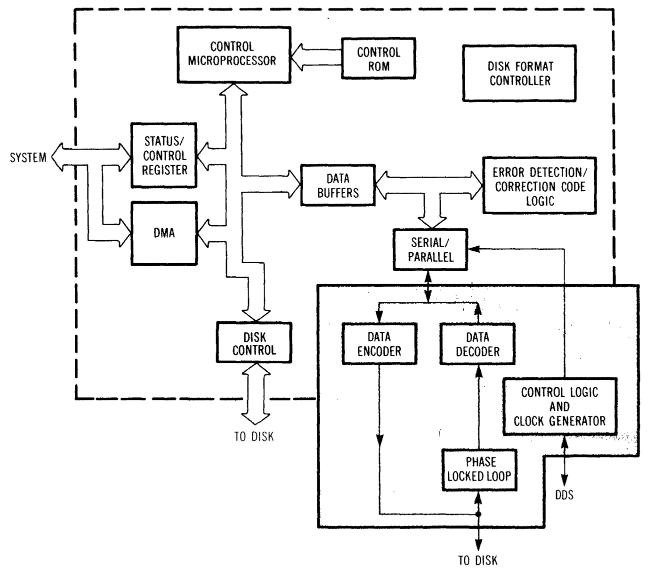


Fig 1 Typical disk controller block diagram shows the partitioning of the hard disk controller (HDC) and the disk data controller in the disk subsystem.

Alternatively, the system designer can customize the controller by adding and deleting features of the chip set with pin straps or software.

The HDC can link the system, via a DMA/CPU interface, to up to four Winchester or floppy disk drives in any combination. On the disk side, the chip's data format controller is fully programmable for disk data and format fields to be accessed. A built-in, 16-bit CPU with powerful software algorithms links the disk and system control units to handle disk control and data transfers. All basic disk controller functions are provided on the HDC: DMA for data transfers, dual-sector buffers for data buffering, Reed-Solomon error detection/correction codes (EDC/ECC), a data format controller, floppy ST506/412HP/custom disk control interfaces, and disk macro commands to assist operating systems in maximizing disk I/O. The companion DDS chip provides a total digital and analog solution to interfacing disks with serial transfer from 125 kbits/s to 15 Mbits/s.

Providing the HDC (or any disk controller logic) with a serial data stream frontend processor gives the DDS separate read write channels. (See Fig 2 for a detailed view of the chip's internal architecture.) In addition, extra control logic supports address mark functions, run-length limited (RLL) codes, and drive select and drive error functions.

The bipolar DDS chip provides stable data during read operations by using an integrated phase locked loop (PLL) to synchronize the disk data for decoding. This improves performance by allowing higher operating frequencies, supplies better noise immunity than discrete implementations, and minimizes costs by eliminating external components.

An onchip, second order, low pass filter within the PLL allows the bandwidth to accurately track the data rate without the noise and cost associated with an external filter. This low pass filter determines the lockup time of the PLL to the raw disk data. Worst case lockup is 2 bytes, regardless of the data rate.

To support data transfer speeds of both floppy and hard disk drives, the PLL handles both the 125- to 500-kbit/s and the 5- to 10-Mbit/s serial transfer speeds. They use the crystal clock to determine the exact frequency. No external components are necessary to adjust the operating speed of the PLL to handle data transfer rates within either the 125-kbit to 1-Mbit/s floppy range or the 4- to 16-Mbit/s Winchester range.

Chip handles Winchesters and floppies

In addition to synchronizing the serial data pulses from the disk, the DDS chip also provides frequency modulation or modified frequency modulation (MFM) decoding. This decoding, in combination with the write channel encoded in the DDS, allows the HDC/DDS chip combination to handle both floppy and Winchester drives without additional logic.

Encoded data/clock pulses are translated into nonreturn to zero (NRZ) data in conjunction with a data reference clock. Disk data formats include special synchronization patterns (address marks) on the disk that allow byte field synchronization for the controller logic. Address mark detection/generation takes the form of a clean control handshake to the

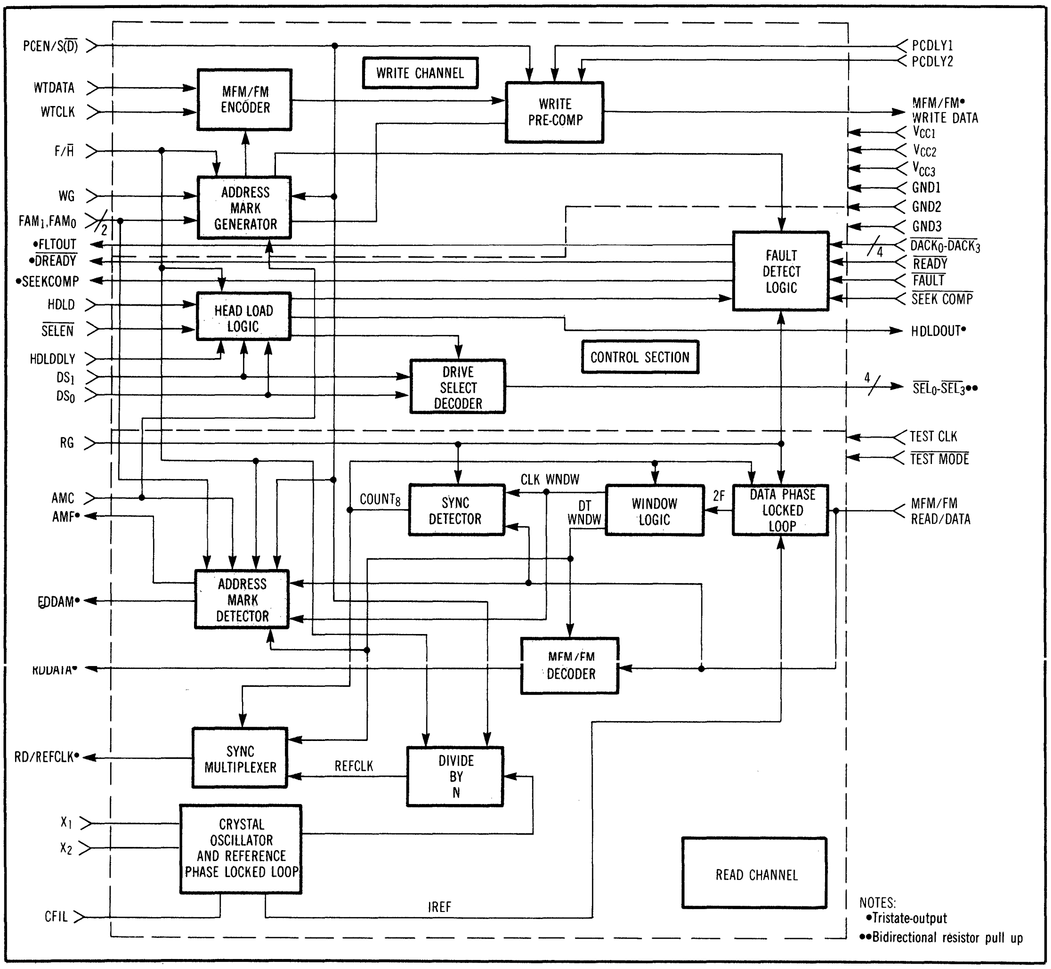


Fig 2 The disk data controller consists of three main sections—read, write, and control. The read section transforms the modified frequency modulation (MFM)/frequency modulation (FM) encoded data into nonreturn to zero (NRZ) data and an associated clock. The write section converts the NRZ data and its reference clock into MFM/FM data to store on the disk. The control section handles the drive select, floppy head load, and fault detection functions.

encode/decode logic. If the HDC is writing the address marks, the address mark control (AMC) line is raised sometime after the write (WG) line. From the next rising edge of the data clock until the address mark found (AMF) line is asserted by the DDS (or custom encoder chip), the HDC waits for an address mark to be written (Fig 3). In the read case, the read (RG) and AMC lines are asserted; assertion of the AMF line by the DDS or custom decoder circuit indicates that an address mark has been found. This signal resynchronizes the controller's internal byte clock for the data read operation. This control sequence allows flexibility in the type of address marks on the disk.

Since the data encoding and decoding function occurs offchip with a simple handshake procedure to generate read and address marks, the HDC permits any type or size address mark. The read channel of the DDS chip provides the address mark detection logic to indicate when such a mark has been encountered. Based on control inputs to the DDS, seven different types of address marks used by Winchester and single/double density floppy disks are recognized. The write channel logic includes the ability to generate these address marks when writing to the disk. A simple handshake between the HDC and DDS indicates when an address mark is written or read.

Precompensation logic built into the DDS solves the problem of magnetic pulse interference that can occur when data bits are packed closely together.

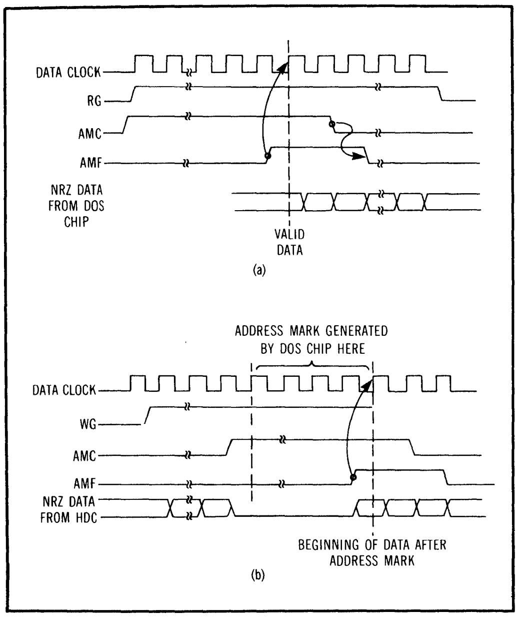


Fig 3 HDC coding is independent of the address mark generator/detector logic. When reading an address mark, the address mark found (AMF) signal indicates when an address mark has been detected (a). Signal AMF indicates that the address mark has been written on the disk, and that the DDS is ready to receive disk data (b).

When pulses are pushed too close together or pulled too far apart (pulse shift), they become more difficult to read. Precompensation methods calculate effects of pulse interaction and compensate for them by "unshifting" data pulses that will be shifted by magnetic pulse interaction before writing pulses to the disk. External delay pins on the DDS (one of the few external components required) allow a choice of shift delay. Depending on the ratio of the resistors attached to these two pins, the precompensation delay value can be selected to within 1 ns or 5 percent of the delay desired, whichever is greater.

Additional functions in the chip eliminate external SSI/MSI "glue" logic. An onchip two to four decoder selects one of four drives. Each of the four drive select pins has a 48-mA driver built into its output, eliminating the need for a separate chip to provide these drivers. Also, since each drive acknowledge input has Schmitt-trigger buffers, extra buffer chips are not required for these pins.

A clock generator within the DDS chip gives the HDC access to a clean clock at all times. Normally, a disk controller that is reading data derives the clock from the variable data/clock encoded in the disk data stream. When writing data, the controller requires a stable frequency at the nominal disk read/write speed. When performing neither of these tasks, it needs a reference clock. The DDS chip automatically switches between these three clocks.

Partitioning disk control functions between the two chips improves the performance of both devices and provides upgrade paths. Putting data encoding/decoding functions on the DDS allows the HDC to handle twice the data rate of VLSI chips that perform the encoding function on the controller. Disk codes such as MFM and RLL require at least twice the normal raw data frequency to handle the encode/decode functions. However, since the high speed bipolar DDS performs this function, the HDC can work with unencoded data rates at up to 20 Mbits/s.

Segmented functions allow flexibility

Keeping the encode/decode function on a separate device also gives the designer flexibility and future upgrade capability. Some interfaces, such as the ST506/412 interface, require that data separation and encoding be performed by the disk controller. Other interfaces, such as the Enhanced Small Device Interface (ESDI), embed these functions in the drive. Thus, putting the encode/decode function in a separate device allows designers to easily interface with alternate standards. Such alternate encoding schemes as RLL can be accommodated by modifying the clock rate and upgrading the DDS chip. (When the DDS is configured with the HDC, the chip set directly supports the ST506/412 and the ST412HP interface, including recovery mode, without any additional logic.)

Segmented much like a board-level disk controller, the HDC uses a 16-bit microprocessor as a front end to the system interface and is coupled with a DMA controller to efficiently transfer data and commands. The format controller is linked to the twin sector buffers, the ECC unit, serial/parallel unit, and a set of user-programmable parameter RAMs. To make basic functions flexible, the HDC provides powerful hardware and software to implement critical areas of the disk controller function. But, it also allows users to selectively disable different parts of the function.

Since the disk data encoding/decoding function is implemented in the bipolar data separator chip, serial data to and from the part is NRZ. This leaves the users free to choose the encoding format. Because a data separator circuit is required, the encoding/decoding function can be put there as easily as in the controller itself. Some interface specifications such as the ESDI standard also require NRZ data to be transmitted over the interface cable. A further benefit of sending NRZ serial data to and from the controller is that higher data rates can be handled without a high speed controller.

The serial data rate is another area in which flexibility is often sacrificed for performance. Most controllers operate in a narrow range of speeds corresponding to the current requirements of the disk interfaces. As technology advances, drive manufacturers naturally want to increase the transfer rates. However, controllers typically cannot handle data rates at transfer speeds beyond those specified in the standards implemented by the board. The HDC adapts to the potential board limitations by providing a wide range of operations.

The HDC's serial channel can support data speeds from 50 kbits/s to 15 Mbits/s. The determining factor is the read/reference clock that is supplied to the HDC. Thus, to raise the data rate on a disk drive, the manufacturer need only change the data reference clock on the HDC. Because the serial format control section is essentially a static design, different speed drives can be attached to the HDC. Whenever a different speed drive is enabled, the HDC handles the drive without delays in switching or overall performance as long as the data reference clock is at the correct speed.

Choice of head positioning technique

A basic disk controller task is to provide the disk functions necessary to select and position the read/write heads over the correct head and track. The HDC supports four head positioning modes by providing implied seek capability in every disk read/write command. If this option is enabled, the difference between the current track/head (which is stored internally) and the desired track head is automatically calculated. Then the read/write heads are repositioned over the new head and track before the current disk read/write command is executed.

Seek overlap can improve disk I/O time in a typical four drive system by as much as 400 percent since all disks can be seeking at once.

A second option allows the HDC to perform simultaneous head positioning operations on different drives, in addition to the normal implied positioning option. A head positioning task (ie, a seek operation) usually requires more time than is necessary to issue the seek command to the disk drive. Once a seek command has been issued, many Winchester drives do not need to talk to the controller until the seek is complete. Thus, after issuing a seek command, the controller can disconnect from the current drive and issue a seek command to the next drive that will need I/O performed. This simultaneous seek operation, called seek overlap, is fully supported by the HDC without additional system software overhead. Seek overlap can improve disk I/O time in a typical four drive system by as much as 400 percent since all disks can be seeking at once. This option still allows embedded seek operations.

Support is also provided for two other disk head positioning methods: restricted mode and buffered mode. In restricted mode, the normal floppy/ST506/412HP step position control is disabled. This allows a head positioning mechanism, such as the command oriented system on the ESDI interface, to be substituted. The HDC still uses the ST506/412HP drive status signals and disk serial data transfer control lines, but no longer performs the actual head positioning. Buffered mode disables all drive positioning controls and status signals, allowing the HDC to be used as a disk data serializer, buffer, ECC, and DMA unit. Additional external disk positioning control logic lets the HDC serve as the heart of customized disk data I/O controller.

Four pins on the HDC are allocated to directly select up to 16 different read/write heads. Internally, the HDC supports up to 256 heads. Additional external control logic accommodates selection of more than 16 heads. Beside head selection, the HDC provides the user with a programmable head settle delay value. A programmable 8-bit value specifies the amount of delay from read/write head selection to read/write head use. This accommodates the variance in head settling time that occurs from drive to drive.

When large amounts of data are read or written using a single command, multiple tracks and heads will be used in the data transaction. However, the specific data arrangement depends on user preference. Some store data in adjacent tracks on one surface; others store all data on the same track, but on adjacent surfaces, so that larger quantities of data can be accessed without moving the read/write heads. To accommodate either arrangement, the HDC automatically processes data over many heads and tracks without system intervention. It supports common head/track options with a user-specified multirecord policy (MRP) for each drive. Possible head/track policies are: move from track to track and change heads only when the surface overflows; move from head to head until all surfaces are used and then switch tracks; or cease operation when the current track overflows and alert the system. The MRP option requires minimum system intervention, even if the tracks overflow during a data transfer.

A reduced write current (RWC) pin on the HDC meets manufacture specifications that tracks in the inner part of the disk surface be written with reduced current in the read/write heads. The HDC allows the user to specify which tracks are affected. The RWC pin can then be properly asserted to reduce the current to the heads. Since these inner tracks are subject to magnetic bit interference, the user can also specify those tracks that require precompensation.

The HDC supports IBM formats for single- and double-density floppies as well as a standard Winchester format. All fields required are programmable in size, pattern, or, in some cases, both. Because of the vast differences between floppies and Winchesters (and between different Winchester drives), all relevant parameters are programmable. Four sets of parameters are kept in the HDC. Whenever a command specifying a particular drive is executed, relevant drive characteristics are taken from the parameter set for that particular drive.

Dealing with data errors

Although disk drives tend to be very reliable in normal use, they are susceptible to errors. Errors result from defects on the disk media, noise in the read/write recovery electronics, or even spurious noise in the data as it is transferred. Therefore, disk controllers incorporate some sort of EDC/ECC logic to protect the data. Four different EDC/ECC options, are supplied by the HDC. The first is the Comitê Consultatif Internationale 16-bit cyclic redundancy check (CRC), an industry standard for floppies. This error-detection-only code makes the HDC compatible with existing floppy controllers and is used to protect ID fields on the disk.

The second and third options are two types of Reed-Solomon (RS) error detecting and correcting codes. The first is a single-burst error correcting code that can detect double-burst errors and correct single-burst errors (Table 1). (A burst is defined as a continuous inversion of the bits in the serial data stream of a given number of bits.) The second RS code is a more powerful version of the single-burst code because it can both identify and correct double-burst errors. In addition to correcting double-burst errors, this double-burst RS code enhances single-burst error correction. More powerful than traditional burst error or fire detection/correction codes (Fig 4), double-burst RS code provides better protection, as MFM type encoding formats yield to more complex, error-prone RLL codes.

Programmable read entry allows error correction before retry, after retry, or after retry fails.

The final option is external, user-defined ECC hardware. When enabled for this option, the HDC turns several pins into status and control signals to control external ECC hardware. Thus, special or custom EDC/ECC codes can be used with the HDC. Many system designers use EDC/ECC codes only for error detection, preferring to perform a reread or simple retry operation to recover bad data. Others combine read retry operations with error correction codes to recover data. A programmable read retry option (up to 16 attempts) on the HDC is coupled with a programmable error correction option. This allows users to specify error correction before each read retry operation, after each retry, or only after all retry operations have failed. Another option halts the HDC whenever a data error is detected, thus allowing the system to intervene.

A special data field recovery command recovers the data field of a given sector regardless of whether its ID field is intact. Normally, if the address marks denoting the beginning of data sector IDs or of the actual data are damaged or destroyed, the controller cannot even try the ECC. As long as the address

TABLE 1

Single- and Double-burst Reed-Solomon Codes

	Sector Size (No. of bytes)	Detection Capability (No. of bits)		Correction Capability (No. of bits)		No. of Check bytes
		Single Bursts	Double Bursts	Single Bursts	Double Bursts	
	128	33	9	9	0	6
Single Burst*	256	33	9	9	0	6
	512	57	17	18	0	9
	128	49	16	25	9	10
Double Burst**	256	49	16	25	9	10
	512	81	24	41	17	15

*Single-burst Reed-Solomon corrects single-burst errors and detects double-burst errors.

**Double-burst Reed-Solomon is an enhanced version of Single-Burst Reed-Solomon; it can detect and correct single- and double-burst errors.

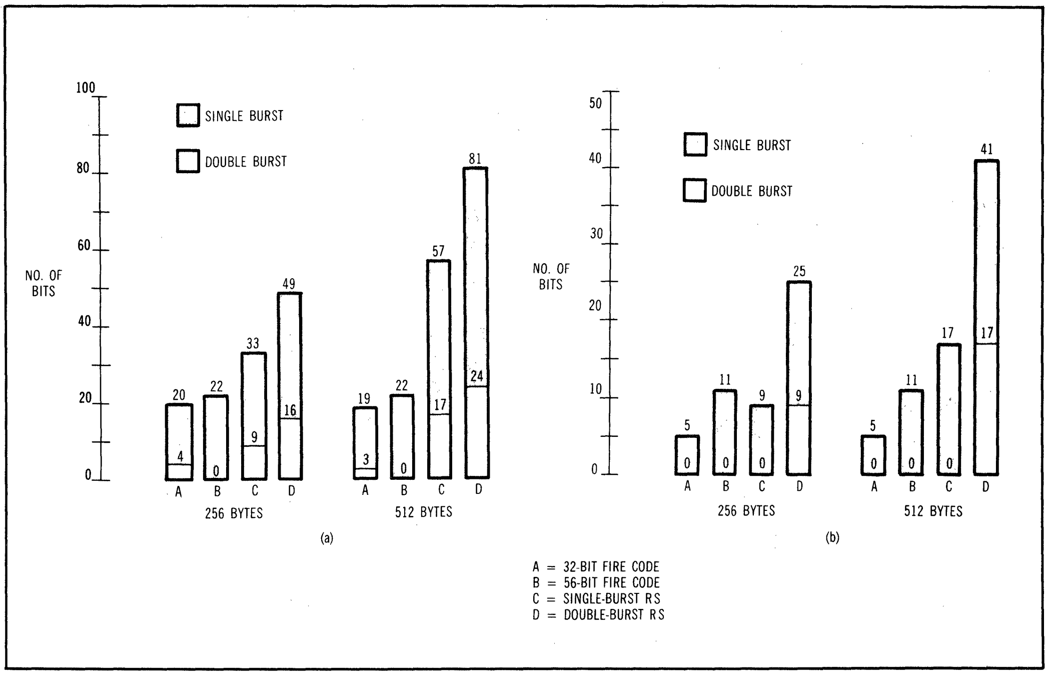


Fig 4 A comparison of the Reed-Solomon (RS) error correction codes versus the older Fire Codes illustrates why RS codes are better. In addition to having better single/double burst capabilities, the RS codes are more likely to detect an error.

mark on the data field is valid, the HDC will read the sector data and if necessary apply error correction to the recovered data.

Because they are mechanical devices, disk drives sometimes have physical defects on the disk recording surfaces. Since fixing the problem can be very expensive, systems usually map around the defect. This makes part of the disk invisible to normal disk accesses.

The HDC supports several methods of defect mapping that require minimal system intervention. The first method relocates sectors from the defective area to another area on the same track. When the system determines exactly where the defect is, the track is reformatted with the defective area mapped out. The system must remember which tracks have fewer sectors, when issuing commands.

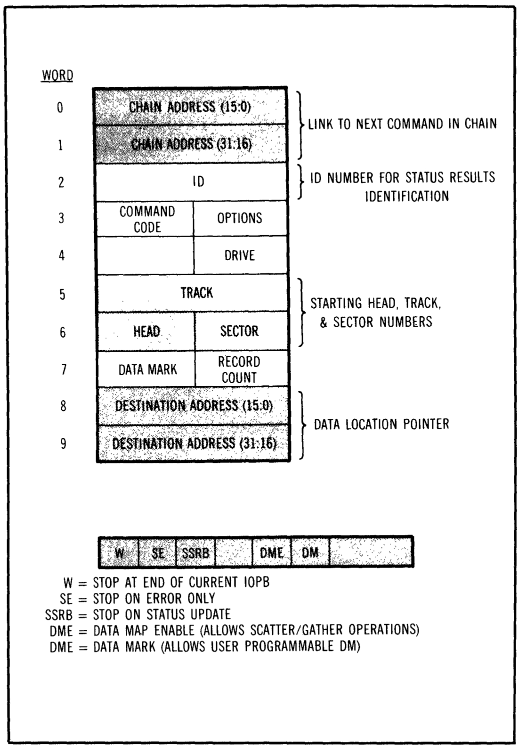
Another method shifts format fields on the track until the defect falls into an unused field. Since field sizes are programmable, they can be altered slightly to ensure that no data or ID information lies in the defective area, once its location and size are known.

The third method of defect mapping relocates the defective track to a defect-free track. A track relocation command provided by the HDC accomplishes this by writing a special vector on the defective track; this vector contains the new location of the track (new head, new track). Then, whenever the HDC accesses the relocated track, it automatically goes to the track's new location to read or write data, if the automatic seek to relocated track option is enabled. No system intervention is required. During multitrack or multisurface read or write operations, the

TABLE 2

Different I/O Parameter Block Commands Available on the Hard Disk Controller

Multisector commands	
(1 to 256 sectors/command)	
	Read
	Write
	Verify
Initialization commands	
	Format (1 Track to entire disk)
	Relocate track
	Restore drive
	Load drive parameters
	Dump drive parameters
Data recovery commands	
	Load buffer
	Dump buffer
	Load syndromes
	Dump syndromes
	Correct buffer
	Read data absolute
General commands	
	Seek
	Read ID



WORD
0
1
2
3
4
5
6
7
8
9
CHAIN ADDRESS (15:0)
CHAIN ADDRESS (31:16)
ID
COMMAND CODE
OPTIONS
DRIVE
TRACK
HEAD
SECTOR
DATA MARK
RECORD COUNT
DESTINATION ADDRESS (15:0)
DESTINATION ADDRESS (31:16)
LINK TO NEXT COMMAND IN CHAIN
ID NUMBER FOR STATUS RESULTS IDENTIFICATION
STARTING HEAD, TRACK, & SECTOR NUMBERS
DATA LOCATION POINTER
W
SE
SSRB
DME
DM
W = STOP AT END OF CURRENT IOPB
SE = STOP ON ERROR ONLY
SSRB = STOP ON STATUS UPDATE
DME = DATA MAP ENABLE (ALLOWS SCATTER/GATHER OPERATIONS)
DME = DATA MARK (ALLOWS USER PROGRAMMABLE DM)

Fig 5 A typical I/O parameter block (IOPB) command illustrates the different command options. For example, a multisector read command would require starting track number, drive, starting sector, number of sectors to be read (record count), and a printer to a block system memory where data is to be stored. Command modifiers allow user options on error response, data mapping, or even user-defined data marks.

HDC can automatically seek to the new track, process the data on the alternate track, and then return to the normal process sequence (in the multitrack/head command) to complete the data transaction.

Controller commands provide extensive options (Table 2). The basic read, write, and verify sector commands allow 1 to 256 sectors to be handled using only one I/O parameter block (IOPB). Overflows to different tracks and heads are handled automatically using programmable options. Options selected for each IOPB modify the method of data transfer to and from system memory, the number of sectors, the starting head/track, error handling, or the type of data mark to be used in the data field (Fig 5).

Embedded software in the HDC determines how the control algorithms allocate internal hardware resources. The HDC's dual sector buffer architecture enables zero-interleave data transfers to run automatically at data rates up to 15 MHz. During a multisector read operation, for example, the HDC always dumps one buffer into system memory while the other buffer is being filled simultaneously with the next sector's data from the disk. Because zero-interleave operation is the natural state of HDC, the time necessary for read, write, or verify operations is automatically reduced two to six times over interleaved operations. However, if sector interleaving is necessary for system considerations, it can be specified. The initial track/sector map is created by the user when the disk is formatted, allowing any interleaving format to be specified. In effect, the HDC maximizes data throughput with a dual-processor architecture. The data format/control processor is active at the same time the macro control processor is calculating information about the next data transfer or sending/receiving data to/from the system via the DMA unit.

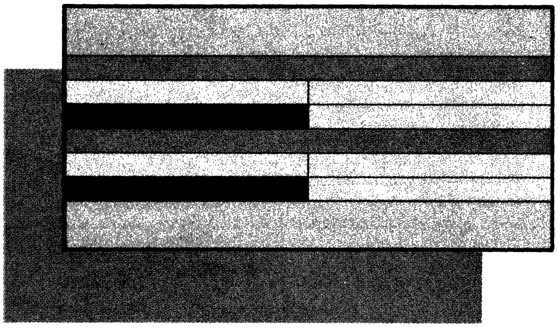
The wide performance range of the Am9580/9581 chip set ensures a long life to disk interface designs. Major chip manufacturers have supported the trend toward utilization of advanced, backward-compatible hardware and software products by providing advanced, higher performance versions of their microprocessor families. The HDC meets this challenge by automatically adapting to oncoming high performance storage devices. A designer need only alter the data separator clock or replace it with a new one to increase the performance of the HDC.

As vertically recorded floppies start arriving on the scene next year, the HDC will be able to handle them without trouble. As RLL encoding replaces MFM encoded data, programming the HDC to use the powerful double-burst Reed-Solomon ECC will maximize data reliability or change data encoding/decoding. As add-on peripherals increase "traffic" on the system bus, the system can alter the HDC's bus characteristics to ease the congestion, or speed bus transactions by raising the system clock to a full 10 MHz. Finally, the ability to customize the HDC interface will allow adoption of proprietary designs, while retaining all the benefits of a powerful, VLSI disk controller nucleus.

WINCHESTER/FLOPPY CONTROLLER EASES DISK INTERFACING

Chip performs seek overlaps and interleaved data transfers for high performance disk control. Other techniques such as disk caching and elevator sorting are easily supported.

by Mark S. Young



Forming a reliable interface between a serial disk and its parallel computer processing data represents the main problem integrators face. Unique requirements of disk/drive system integration such as disk control, data transfer, data reliability, and error handling demand costly hardware and extensive software support. Also, coupling an asynchronous, serial data channel to a synchronous, parallel computer data channel can prove to be a difficult hardware task. Data buffering, data blocking/deblocking, bus usage, and system interface timing require extra system logic. Finally, making the disk conform to the operating system's data handling needs requires development of additional software.

Creating disk directories, addressing disk memory, optimizing disk access, and using the controller disk I/O commands all combine to make interfacing difficult. One chip set that addresses these concerns is the Am9580 hard disk controller (a Winchester/floppy controller) and the Am9581 disk data separator. The set is ST506/412 compatible, but it can be made to conform to Enhanced Small Disk Interface or Small Computer System Interface by adding several other chips.

One of the disk controller's primary tasks is coupling the serial disk bus from the parallel system bus. Even though such peripherals as modems and printers allow the user to stop a transfer after each character by using a ready/not ready signal line, halting a disk is not as easy. The nature of disks prevents such a data transfer interface; once it starts transmitting data from a sector, the operation must either finish or abort. Because data transfer cannot be stopped, the controller and the system must deal with complex data buffering and transfer over the system bus.

A typical Winchester drive sends data serially at 5 Mbits/s. If the data is moved directly to the system bus in 16-bit blocks, the disk controller must hold the bus for more than 800 μs to complete a 512-byte transfer. Because data leaves the disk at a relatively

Mark S. Young is a product planning engineer in charge of the hard disk controller chip at Advanced Micro Devices, Inc, 901 Thompson Pl, Sunnyvale, CA 94086. He holds a BA in computer science from the University of California at Berkeley.

slow speed (but not so slow that the bus can be shared among different masters), the disk controller can easily become a system bottleneck. This is especially true in multi-user or multitasking environments.

Several different data buffering methods are commonly used to prevent this bottleneck. The first in, first out (FIFO) buffer provides a simple, and generally effective, method of buffering disk data in low performance systems that use floppy disk drives. In the higher data speed environments of Winchester drives, however, a FIFO does not provide sufficient bus buffering. Moreover, the FIFO does not free the bus from the problem of long accesses by the disk controller—it only decouples the disk from the CPU bus timings. If the FIFO is not sufficiently deep (256 bytes or more), the bus buffering only provides longer lag between the time disk transfer begins and the time it takes to obtains access to the bus through arbitration. In the mean time, the controller still needs to use the bus for long periods.

Buffer dilemmas

A FIFO buffer may also encounter data underflow and overflow. These problems are especially prevalent in systems with high bus loading or with multiple bus masters that require frequent servicing. Usually, if another peripheral interrupts a disk controller during a data transfer, this operation must be aborted and tried again. FIFO buffering cannot prevent this event, unless the depth of the FIFO is sufficient to store an entire sector. Finally, the FIFO does not allow the controller to determine the validity of the data—that is, whether the data has been certified as correct by the error detection circuitry—until most of the sector has already been sent to the system. Either incorrect data must be corrected using additional bus cycles, or the entire sector must be reread from the disk and transferred to the system again.

A full sector buffer is another technique that is commonly used in Winchester controller designs. The single sector buffer normally uses a single static RAM chip. The size of the buffer is limited only by the SRAM devices available. With 2-Kbit x 8 SRAMs widely available today, and the newer 8-Kbit x 8 SRAMs starting to appear, just about any size buffer can be easily built. Multiplexing the address, data, and control lines between the controller and the system requires a minimum of extra hardware with this buffer.

The full sector buffer approach avoids many of the problems associated with the FIFO approach. Overflow and underflow are no longer a problem, and data integrity is ensured, because the entire sector is stored in the buffer before it is sent out to the system. However, the single sector buffer cannot perform zero interleaved data transfers between the system and the disk controller—data cannot be loaded into the buffer until the previous sector has been fully emptied. As a result, single buffer disk I/O can be at least 25 percent slower than zero interleaved sector operations, depending on how much interleaving is required.

The dual sector buffer approach used on the hard disk controller has the advantages of single sector buffering, while it greatly increases disk I/O throughput. These integrated dual sector buffers allow the user to select data sector sizes of 128, 256, or 512 bytes. The dual sector buffer system employed by the hard disk controller always has one buffer available to the system and the other buffer available to the disk [Fig 1(a)].

When both the system and the disk are finished using their respective sector buffers, the buffers are switched (or toggled) so that the former disk buffer is now available to the system and the former system buffer is available to the disk [Fig 1(b)]. This buffer toggling method allows the hard disk controller to transfer data continuously from the disk without any sector interleaving, even at disk speeds of 20 Mbits/s. Besides allowing complete decoupling of the disk and the system bus from each other, the dual sector method maximizes data throughput as it minimizes system bus use. Because each data sector on the disk is stored entirely within the hard disk controller, the

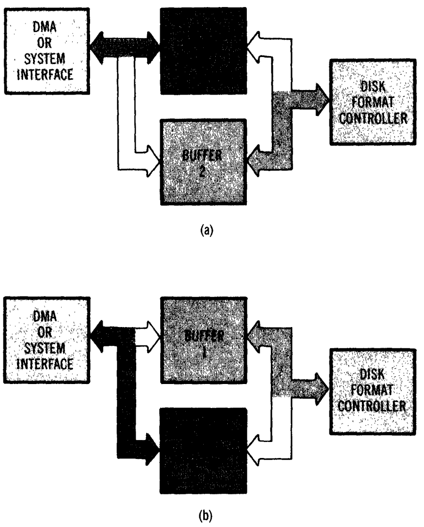


Fig 1 The dual sector buffer approach illustrates how data can be transferred between the disk and the system simultaneously. Buffer No. 1 is allocated to the system/DMA while the other buffer is used by the disk formatter (a). When the system and formatter are finished, the buffers are switched (b). In this manner, the system empties one buffer while the disk formatter fills the other (or vice versa), thus continuously moving data to and from the disk.

system bus is not tied up continuously during the physical transfer of the data to and from the disk.

The controller's bus use can be limited to short, fast data transfers without the danger of data underflow or overflow. In addition, the controller can be interrupted by more important activities for much longer periods than allowed by FIFOs, without sacrificing any disk data transfers. And because the sector is fully loaded onto the controller and checked for errors before it is transferred to system memory, the data need only be sent once. Erroneous data is never transferred out of the controller unless the system specifically requests the controller to do so. Full sector buffers allow the Winchester/floppy controller chip to gather from diverse memory locations on-the-fly, instead of assembling contiguous data blocks before the transfer occurs.

To adjust overall bus performance, the system can alter burst and dwell values on the hard disk controller.

All disk controllers, especially Winchester controllers, must perform data transfers efficiently between the disk and system memory. Usually, this requires that a DMA device or DMA channel be dedicated to the disk controller. From a design viewpoint, a compromise is made when an external DMA controller operates with a disk controller. This is because an external DMA controller is usually slower than a custom, integrated DMA unit.

In order to make the DMA control handshake work properly, the DMA must receive high priority in the system bus hierarchy. At the same time, the interface must be generalized to handle the different DMA devices. Extra software is also required by the system to set up the DMA controller for each disk transfer. From a controller design point of view, interfacing to an external DMA unit requires a tradeoff in disk controller performance. If the controller has to handle commands, status, and data without constant system intervention, at least three DMA channels have to be dedicated to the disk controller.

The fully integrated DMA controller on the hard disk controller eliminates all external system hardware support functions. Thus, system interface requirements are reduced to bus drivers, bus receivers, and a disk driver software routine. The Winchester/floppy controller chip's DMA controller can handle an 8- or 16-bit data bus and up to 32 bits of addressing. The hard disk controller's DMA unit can transfer data at up to 5 Mbytes/s.

An extra strap option allows the system interface to support either synchronous or asynchronous memory transfers. The user can also extend the basic four-clock transfer cycle with either the READY line or with programmable wait-state insertion. A failsafe measure ensures that a faulty memory board will not cause the hard disk controller to stall the system bus. A time-out counter releases the system bus, thus ensuring that the hard disk controller aborts any memory transaction taking longer than 5 ms.

The Winchester/floppy controller's DMA bus activity can be determined by setting the burst and dwell characteristics in one of its control registers. The burst characteristics indicate how many bytes of data are transferred every time the DMA unit uses the system bus. This number can be programmed from 1 byte to as many bytes as are required in a single transfer request of the hard disk controller. The dwell characteristics determine the amount of time the DMA unit stays off the system bus between bursts. During a multibyte DMA transfer, the DMA unit will release the bus. This occurs only after the number of bytes indicated by the burst count have been transferred. After waiting the amount of time specified in the dwell count, the hard disk controller will request the bus to continue the DMA transfer, as necessary.

To adjust the overall system bus performance, the system can alter burst and dwell values on the hard disk controller. These burst and dwell characteristics allow other peripherals in the system to use the bus and to prevent "lockout" of lower priority peripherals. Because the DMA unit can transfer data to and from the buffers much faster than the disk can, the onboard sector buffers give the system plenty of time to move data in and out of the controller. The buffer will maintain maximum disk throughput, even with the use of burst and dwell.

Controlling the bus

In the event of a system emergency, the DMA unit can be forced to abort the current DMA transfer. A two-wire handshake scheme is used on the Winchester/floppy controller chip interface to gain control of the system bus. The hard disk controller asserts the bus request line and waits for the bus acknowledge (bus grant line) to be asserted by the system arbiter. When the DMA unit is finished transferring data on the bus, it de-asserts the bus request line. This indicates to the arbiter that the bus is now free. If the system needs to force the hard disk controller off the system bus, all it has to do is de-assert the bus acknowledge line. The hard disk controller will cleanly finish the current word or byte transfer, and then get off the bus. This is called preemption. Even in an emergency, none of the data transfers to or from the hard disk controller are affected. The hard disk controller will continue to request the bus as it would normally and attempt to finish the interrupted transfer as though nothing unusual happened.

The system has access to seven 16-bit control/status registers, as well as to an onboard DMA unit.

These registers allow the system to keep track of the hard disk controller while the chip is operating. The command/status register allows the system to start, stop, or resume a command chain operation. A software "reset" is also provided. Its function is equivalent to that of the hardware reset pin. Software reset allows users to reset the chip without actually using the reset pin, which can be tied to the system reset line. This causes the chip to immediately abort the current command operation.

The system has access to seven 16-bit control/status registers, as well as to an onboard DMA unit.

The stop chain command forces the hard disk controller to abort the current chain at the end of the current command. The resume command allows the hard disk controller to pick up where the interrupted command left off without restarting the command. This is possible when a noncatastrophic error is the cause of the interruption. The upper byte in the register contains the current chip status—information about certain types of errors are reported in this field whenever the hard disk controller halts.

The mode register determines bus activity by using the burst or dwell fields. In addition, a programmable interrupt option allows users to enable or disable interrupts from the hard disk controller. The number of wait states inserted into each memory transfer (0, 1, 2, or 3) can be specified to allow the hard disk controller to accommodate slow memories and the memory transfer acknowledge line. A register lock bit freezes the register set. This allows the system to monitor all the registers while the hard disk controller is running, without the danger of having them altered during a read access. Finally, the disk control interface is specified in this register by the seek mode field. It determines whether implied and overlapped seeks, implied seeks only, no seek operations but disk status (restricted), or no seeks or disk status (buffer mode) will be performed by the hard disk controller.

The next block pointer always points to the current command being executed in the command list. The status result pointer always points to the next status block to be written. The status result length specifies how much memory is allocated for status results information. The next block pointer is initialized before each go command; it points to the beginning of the command chain. Then, the hard disk controller automatically fetches commands and reports their status to the area specified by the status result pointer and status result length registers. The status result pointer and status result length registers only need to be updated when they overflow. By monitoring these registers, the system can determine what the hard disk controller is doing while this chip is executing a command list. The next block pointer and status result pointer are both 32-bit pointers, accessed as two 16-bit registers.

Among the most poorly implemented features on many disk controllers are the command set, status information, and the mechanism for transferring these between the system and the controller. The small, primitive disk I/O commands in many controllers force the system to build up more powerful macro commands just to facilitate the writing of the disk driver software routines.

A typical controller might have read, write, and verify sector commands. It would also have a seek, a load/dump buffer, a format track command, read error correction code syndromes, and read ID. These primitive commands usually require a separate seek command before any track can be accessed. They also require the system to monitor the position of the read/write heads. In many controllers, the drive parameters must be loaded every time the drive is accessed, or the flexibility of the user is limited by fixed drive control parameters. Additionally, the system must continuously feed all commands to the controller, keep track of errors, and control the data transfers to and from the disk.

An indirect software cost associated with a simple disk controller design might result from interfacing a typical Winchester with a personal computer. This would require a complex software I/O driver to

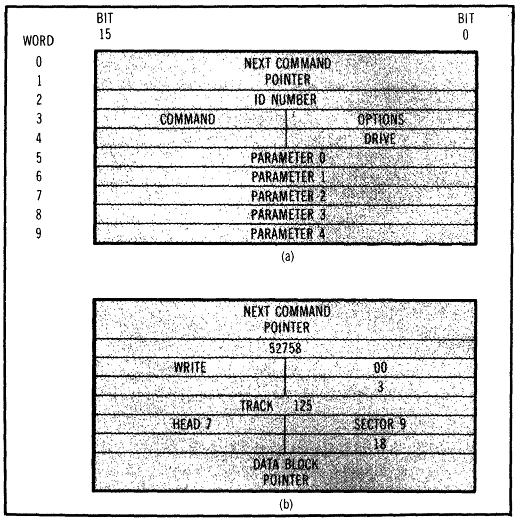


Fig 2 The I/O parameter block (IOPB) command structure is a 10-word block containing all the data needed to allow the hard disk controller to move data to and from the disk (a). The write command will transfer 18 sectors from a contiguous block of system memory indicated by the data block pointer (b). The read/write heads will be moved to track 125, head 7, and start in sector 9.

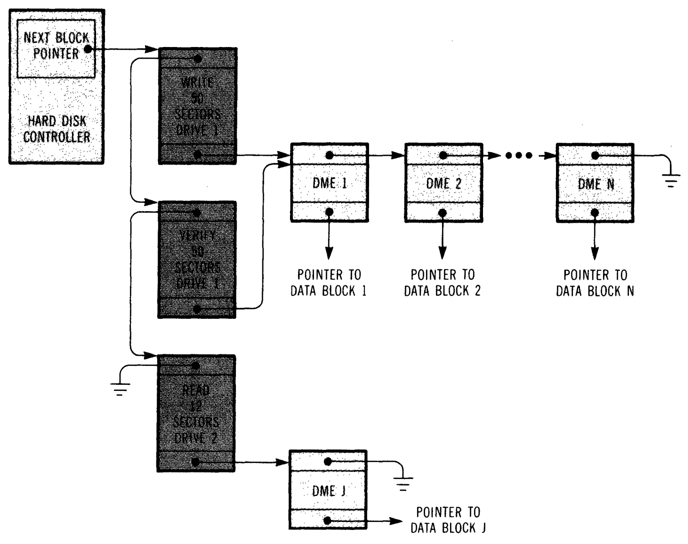


Fig 3 The IOPB command list is a linked list structure residing in system memory. The linked list structure facilitates command list manipulation. Optional data map entries (DMEs) allow data blocks of any size or memory location to move to and from the disk. The hard disk controller automatically fetches and processes the command list without any host CPU intervention.

perform simple disk read and write operations. For example, when an operating system requests 10 disk sectors, the CPU must issue a seek command and 10 groups of alternating read sector and dump buffer commands. Besides issuing these 21 commands to the controller, the external DMA controller must be set up for the data transfer. At the same time, the CPU becomes responsible for monitoring the controller's progress after each of these commands has been completed. The hard disk controller, on the other hand, could complete such an operation in a single command and alert the system via interrupts if any problems requiring intervention of the operating system arose.

The basic command structure of the hard disk controller is the I/O parameter block (IOPB), shown in Fig 2. All disk-specific parameters are obtained from the individual drive parameter RAMs that are stored on the hard disk controller. Only parameters for executing the command are required. The first two words of the IOPB contain the 32-bit pointer to the next IOPB in the command chain; the IOPB equals 0 when it is the last command in the chain. Finally, a 16-bit ID value is provided to help the system identify errors.

When the hard disk controller issues status result block commands, the ID field of the IOPB is put into the status result block to indicate which IOPB caused the error or had a problem. The command byte says what command the IOPB gave. The operating system has 16 different IOPB commands. The options byte allows users to specify both command-independent and special command-dependent control options. The command-independent options allow users to stop on any status result block, on fatal errors only, or at the end of each IOPB. The command-dependent options include track verify after a seek, data map control enable, user-defined data marks, error location information, and location of specific disk ID fields.

Up to six 16-bit parameters are provided with each IOPB. Typically, the drive's starting track number, starting head number, starting sector number, and the number of sectors to be transferred are required in each command. In addition, options such as user-defined data marks, format patterns to be used, and relocated track information are needed. A 32-bit data pointer is also needed to specify the beginning of the data to be transferred, or to point to the data map indicating how the data is transferred.

Each of the controller's primary commands—read sector, write sector, and verify sector—can handle from 1 to 256 sectors. Specific user-programmable bytes in the drive parameter RAMs indicate how the controller will respond to multihead and multitrack overflows. All commands on the hard disk controller automatically seek the track specified in the parameters and perform any multiple track seeks the command requires. A separate seek command is not required for any disk I/O command. Each command also has a pointer or pointers to the data needed for write or verify operations, including pointer or pointers to memory space in the case of read operations. This allows the hard disk controller to perform all buffer read and write operations automatically between the controller and system memory without using separate load and dump buffer commands.

Other commands such as load disk parameters, restore drive, and format track allow the user to initiate the controller for normal operation. The format track command specifies from 1 to 65,535 tracks. It also formats the entire disk with one command, again using the mutlitrack, multihead policy byte to handle overflow. Once the disk is initialized, the hard disk controller monitors the location of the current read/write heads' tracks, thus relieving the system of the responsibility. Moreover, a seek command allows users to move the disk read/write heads to any track. This is useful, for example, if a disk is to be accesssed in the near future.

The read data absolute and read ID commands enable recovery of data from disk fields that are damaged by head crashes or media defects. The relocate track command allows the user to move data from defective tracks and re-map the new track onto the old track. Whenever the damaged (relocated) track is accessed, the hard disk controller will detect this. It automatically moves to the new track without stopping the command or burdening the operating system.

The dump sector buffer, dump error correction code syndromes, and correct buffer commands allow the system to examine disk data error before correction. If necessary, the hard disk controller can perform error correction automatically, without system intervention. The load sector buffer, load error correction code syndromes, and dump drive parameter commands, in conjunction with the other load and dump command, allow onchip diagnostics of the Winchester/floppy controller chip's degree of function.

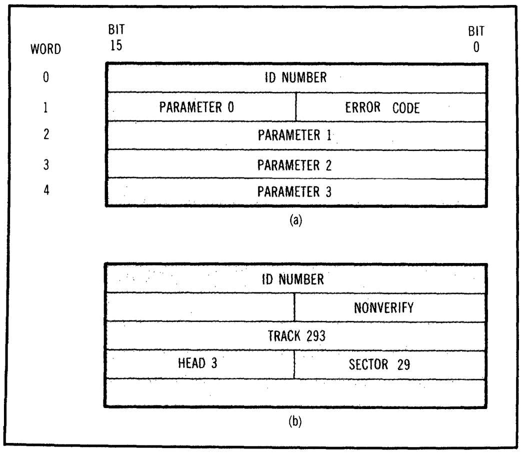


Fig 4 The status result block reports complicated disk I/O errors to the system. Each five word status result block has an ID value and up to four parameters (a). A typical status result block illustrates the type of status information given to the CPU (b). In this case, the data specified by a verify command was not the same as data found on the disk's track 293, head 3, and sector 29. The drive number is specified in the IOPB with the appropriate ID number.

With many disk controllers, the operating system must issue commands to the hard disk controller. The hard disk controller fetches its own command instead. The operating system arranges the command in a linked list command chain (Fig 3). When the next block pointer points to the beginning of the list, the hard disk controller is ordered to execute the command chain. Then the hard disk controller fetches the commands, processes them, and transfers the data to and from the disk and system without system intervention. Using the link pointer in command will cause the hard disk controller to fetch commands until the last command in the chain is executed.

Operating system stays informed

When the hard disk controller completes the command chain, or when an error/control fault occurs, the hard disk controller interrupts the system to tell it what has happened. If a correctable error or fault caused the command execution, the system tells the hard disk controller to resume processing, but instructs it not to restart the command. Other options allow the hard disk controller to interrupt the operating system whenever data is transferred and is therefore available.

The linked list command structure eases the implementation of the overlapped seek routine. A special software routine handles the command chain in the hard disk controller chip. When the operating system enables this routine, the hard disk controller follows the command chain via the linked pointers, and looks for commands ahead of the command being executed. Whenever it finds a command involving an inactive drive, it calculates the necessary seek information. It then issues the seek command to this drive, well before the command is actually executed. And when a previously busy drive finishes its current command, the hard disk controller starts to look farther down the chain for a command that involves some seeking for that drive. In this manner, time-consuming head positioning is executed in parallel, providing a dramatic improvement in disk I/O performance.

In addition to fetching and processing its own commands, the hard disk controller issues status result blocks to indicate when certain events occurred while disk data was being processed. The status result block is a five-word unit (Fig 4). Each block contains a 16-bit ID number to say which IOPB issued the status result block, an error code to explain the error, and up to 7 bytes of parameters to indicate other error-specific information. For memory efficiency, status result blocks, unlike IOPBs, are stored in consecutive order in the status result area specified by the status result length register.

By examining the status result area after a command chain has been executed (or halted because of

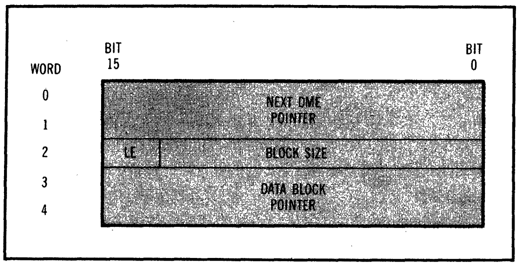


Fig 5 The DME uses a linked list structure similar to the IOPB command list. Each DME can handle up to 32,768 bytes. The load enable (LE) bit determines whether the data specified by the DME will be skipped. This option specifies what data in each sector will be downloaded, thus avoiding unnecessary data transfer on the system bus.

errors), the system can determine what has happened on the disk. When the status result area becomes full or overflows during a normal hard disk controller operation, the hard disk controller ceases operation and alerts the system. Then, the system can either increase the area allocated for status reporting or analyze the status and reset the status result pointer and status result length to the beginning of the status area. It performs one of these functions before telling the hard disk controller to resume operation.

Optional stops

Status reporting is an important function of a peripheral that controls the system's primary mass storage device. Many errors can occur in a typical disk subsystem during normal processing; each affects the system, controller, or disk to a different extent. For example, upon detecting an incorrect seek operation, the hard disk controller will automatically attempt a re-seek operation. If the second attempt is successful, the hard disk controller issues a nonfatal status result block to alert the system of the disk error. If the reseek attempt fails, however, the hard disk controller automatically issues a status result clock and ceases operations. The system should have the option of specifying whether or not the controller will stop on any error (fatal or not) or only on the fatal errors.

It may be unwise to abort a disk operation even for a fatal error in some cases. Consider a zero interleaved multisector read operation. In the process of transferring the disk data to the system, a flaw might be noticed by the error detection unit. Because few error correction codes can correct the error and physically continue reading the sectors in sequence, a typical controller would abort the read operation and immediately initiate data recovery operations. However, since data errors are relatively rare in most drives, it would be wiser for the controller to indicate the sector that is damaged, prevent the error from being transferred to the system, and continue reading the remaining sectors. Once the last of the data has been transferred, the system has the controller recover sectors that had problems. In this scenario, the controller avoids wasting time caused by stopping and starting the read procedure or resulting from introduction of extra rotational latencies into the disk I/O operation.

The status result system in the hard disk controller handles errors in many different ways. Users are given maximum flexibility in deciding what errors are to be aborted automatically and which errors are merely informational. Users of most small personal computers probably do not want to halt the controller every time an error is detected—especially if the controller can be programmed to perform retry operations automatically and to use the onboard error correction codes. These users are only interested in being alerted when the data cannot be recovered automatically. Users of minicomputer systems might want to examine the status result information for system maintenance logging. Users of such error-sensitive systems as fault-tolerant computers have a greater investment in detecting errors, no matter how minor. The hard disk controller allows these users to tailor the status system to support their applications.

Data in a computer system's memory is represented by groupings of bytes or words of various sizes that, on a disk, are divided into groups of bytes called sectors. Although sector sizes vary from disk to disk, and sometimes from track to track, they are always the same size on a given track. Sectors on a disk usually come in 128, 256, or 512 bytes; less frequently in 1024, 2048, or 4096 bytes. Unfortunately, computer data and disk data rarely agree on sizes. The controller must usually divide all system data into blocks equal to the sector size on the disk. The reverse operation is required when data is read off the disk.

The embedded control algorithms in the Winchester/floppy controller chip allow this common data preprocessing task to be offloaded onto the controller. The hard disk can automatically fit data into sector blocks for writes and into continuous data for read operations. It can also perform data blocking and deblocking for any other size of data.

The hard disk controller allows the user to specify an optional data map whenever disk data is transferred between the disk and system memory. With this data map, users can fetch data from the system memory in any size block (1 to 32 Kbytes/block) and pack the data into the required number of sectors. Conversely, when data is being read from the disk, the data mapping option allows users to deblock the data into any size block and store it in widely dispersed, noncontiguous memory locations. Users also have the option of reading only certain fields from each sector and transferring only that data into the system. Thus, bus bandwidth is saved from unnecessary data transfers.

Data in system memory is not always arranged in continuous blocks that are easily moved, as units, to the controller and onto the disk, however. Increasingly, users are creating complex data structures in Pascal, C, and Ada to store data in a form that can be easily manipulated. It is very inefficient to store data in blocks of memory that encompass the data structures themselves. It is also wasteful to have the CPU gather the data and transfer it to a temporary buffer for the disk controller. The same applies when the data needs to be read back into the system. Having the CPU perform the data packing and unpacking requires extra system memory for buffers and increased system overhead because all data needs to be transferred twice in system memory to be read from, or written to, the disk.

The data map structure used by the hard disk controller is a block of five 16-bit words (Fig 5). It consists of a 32-bit pointer to the next data map block, a byte count word with a load flag, and a 32-bit pointer to the actual system memory location where data is to be read or written. Data map blocks can be strung together just like IOPBs to create a data map as large as needed for the data blocking/deblocking task. Because the DMA unit is much better at handling data transfers than the CPU, the data map is considerably more efficient than having the system perform the task.

As disk controller peripherals become more sophisticated, the system interface must also progress. Although it is impractical to provide users with every permutation in hardware and software, the hard disk controller design allows for a large degree of flexibility and programmability. At the same time, the disk controller should not overwhelm the system with a design so flexible that it is unusable. System integrators need a controller that can be easily integrated into today's standards and upgraded to future requirements without complete redesigns or compromises in performance.

64-kbit EEPROM speeds in-system reprogramming, adds data polling

Besides having a large capacity, an electrically erasable chip writes both by page and by the byte and inverts data to indicate when it is writing.

For program storage, designers of microcomputer systems originally had to rely on ROM. If they wanted the advantages of erasability, they sometimes used battery-backed RAM. Not only is this approach expensive; in addition, the batteries must be hand- rather than wave-soldered and they will eventually run down, requiring that they be replaced.

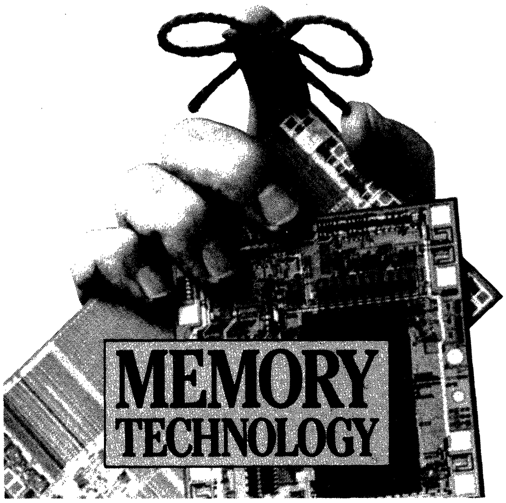
ROMs evolved into EPROMs, but EPROMs also have their drawbacks: They must be physically removed from the system, irradiated with an ultraviolet lamp for 15 to 20 minutes, programmed for another 3 minutes, and then returned to their sockets. That amounts to some 30 minutes wasted. Furthermore, leads can be bent, and dust and foreign objects can get into open equipment.

EEPROMs, which first became available 12 to 18 months ago in 5-V-only form, do not present the problems that EPROMs do, since they can be reprogrammed while still in the system. Also, their packaging costs are lower because no window is needed. As the technology advances, the differences in die cost will disappear, and EEPROMs will be less expensive than the other options. The availability of affordable large-capacity EEPROMs will help to expand various applications—in medical equipment, for example, or workstations—and open up new ones like robotics.

One of the first EEPROMs with a capacity of 64 kbits has two special features—page-mode writing and data polling (in addition to ready/busy signaling). Page-mode writing is extremely useful for designs that require fast

Ken Pope, Advanced Micro Devices Inc.

Kendall W. Pope received a BSEE from the California Polytechnic Institute in San Luis Obispo. Before joining AMD, he worked on test equipment design at Intersil and on application engineering for static RAMs at Synertek. Since 1983, he has been involved in applications and technical marketing with AMD's nonvolatile memory group.



writes. Data polling allows a single read and comparison operation to determine the status of the chip, eliminating the need for external hardware with this function. In addition, since larger-capacity EEPROMs will have to sacrifice their Ready/Busy pin for greater addressing and turn to data polling, designs employing the chip will be easily upgradable.

The device, designated the Am2864A, is completely self-timed, leaving the processor free to perform other tasks until the Ready/Busy signal or data polling indicates that writing is complete. It runs on a single 5-V power supply.

Reading and writing

The 2864A can be described in terms of its reading and writing modes. The reading mode features automatic selection, a 200-ns access time, 75-mA active and 25-mA standby current drain, and unlimited read cycles.

The writing mode features 5-V-only programming, a self-timed 10-ms erase/write cycle, automatic erasure before writing, 32-byte page writing, latched addresses and data, and power-failure write protection circuitry, as well as a Ready/Busy signal and data polling.

Automatic selection allows the reading of a binary code from the device that will identify the manufacturer and memory tape. This feature is intended for use by programming equipment so that the device can be automatically matched with the corresponding programming algorithm.

Page-mode writing allows 1 to 32 bytes of data to be written into the EEPROM in a single write cycle. The effective writing rate is 30 μs/byte, or 2.6 seconds to rewrite the entire chip. By having internal latches for addresses and data, the part eliminates all the external logic required to perform data writes.

Solid protection

Three features protect the device from being inadvertently written into. To avoid the initiation of a write cycle while the supply voltage (V_{CC}) powers up or down; a write cycle is locked out if V_{CC} is less than 3.8 V. To protect against noise, a write-enable lockout circuit prevents Write Enable ($\overline{WE}$) pulses of less than 20-ns duration from initiating a write cycle. Lastly, a write cycle cannot be initiated when $\overline{WE}$ is high, Output Enable ($\overline{OE}$) is low, or Chip Enable ($\overline{CE}$) is high.

The Ready/Busy signal (R/$\overline{B}$) is actively driven to a logic low (V_{OL}) while the device is performing a write cycle. This output is an

Table 1. Mode and feature selection

	Inputs			Outputs		
Mode or feature	$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	R/$\overline{B}$	I/O	A_9
Reading	Low	Low	High	High	Data Out	X
Writing	Low	High	⎺⎽⎺	⎽⎺	Data In	X
Standby	High	X	X	High	High impedance	X
Read/inhibition	Low	High	High	High	High impedance	X
Write inhibition	Low	Low	⎺⎽⎺	High	High impedance	X
Automatic selection	Low	Low	High	High	Code	V_H
Data polling	Low	Low	High	Low	$\overline{\text{Data In}}$	X

X = don't care, V_H = 12.0 ± 0.5 V

open-drain configuration to allow several EEPROMs to be connected in a wired-OR fashion. R/$\overline{B}$ will be in a high-impedance state once the write cycle is complete, indicating that the device is available for another operation.

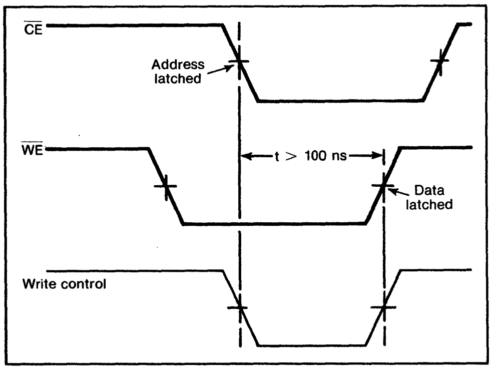
The device is selected when $\overline{CE}$ is at the low voltage level (V_{IL}); when this input is at the high voltage level (V_{IH}), the device is in the standby mode (Table 1) and the device's power consumption is reduced by 70%. Once a write cycle has been initiated, the device remains active until the cycle is completed. $\overline{CE}$ should be kept high during the writing mode, unless the data polling mode is being used as well.

Data is written into and read from the device through the I/O pins. These pins are in the high-impedance state when either $\overline{CE}$ or $\overline{OE}$ is at V_{IH}.

The $\overline{OE}$ pin controls the I/O buffers. When it is at V_{IH}, the output buffers are in the high-impedance state. If a $\overline{WE}$ pulse is applied to the device while $\overline{OE}$ is at V_{IH}, the I/O pins become inputs.

Once in a write cycle, $\overline{OE}$ is in a "don't care" state unless the data polling mode is selected, in which case $\overline{OE}$ must be at V_{IL}. Data is read from the device when both $\overline{CE}$ and $\overline{OE}$ are at V_{IL} and $\overline{WE}$ is at V_{IH}.

The 2864A features several enhancements to the write operation. A byte write cycle is initiated by holding $\overline{OE}$ high and transitioning $\overline{WE}$ while $\overline{CE}$ is held low, or switching $\overline{CE}$ while $\overline{WE}$ is held low, or a combination of the two. Whatever falling edge occurs last initiates the write cycle and latches the address. Whichever rising edge occurs first latches the data and control signals. It is possible to mix $\overline{CE}$ and $\overline{WE}$ to accommodate different microprocessors, as long as both signals are simultaneously low for a minimum of 100 ns.



1. A byte write cycle for Am2684A 64k EEPROM is initiated by holding Output Enable high and transitioning Write Enable with Chip Enable held low or switching $\overline{CE}$ with $\overline{WE}$ held low, or a combination of the two. Whichever falling edge occurs last initiates the write cycle and latches the addresses. Whichever rising edge occurs first latches the data and control signals.

The possible combinations of $\overline{WE}$ and $\overline{CE}$ are referred to as write control. The falling edge of write control is the later transition of either $\overline{CE}$ or $\overline{WE}$, and the rising edge of write control is the early transition of $\overline{WE}$ or $\overline{CE}$ (Fig. 1).

The write cycle consists of the erasure and then the programming of a byte. The write cycle time is internally specified at a maximum of 10 ms, and all write controls, voltages, and timing are internally generated once a write cycle is initiated. The R/$\overline{B}$ output will be low (indicating that the chip is busy) while the part is executing a write cycle. The chip can be deselected as long as R/$\overline{B}$ is low, to allow the processor to perform other tasks.

If the device is read ($\overline{WE} \geq V_{IH}$) while in the write cycle, the complement of the data being written will appear at the outputs. This is the data polling feature.

Writing by the page

As noted, automatic page writing allows 1 to 32 bytes of data to be written into a single page in a single operation. Sequential write control pulses load the Y address and the data for each byte into a 32-byte address and data buffer. Just as with byte writing, the Y address is latched on the falling edge of write control, and the data is latched on the rising edge. The page address is latched on the falling edge of write control during the last byte to be written.

The rising edge of each pulse produced by write control will turn on a 300-μs on-chip timer. The next word of the page may be loaded within this period. The write operation will begin once the write timer has been allowed to time out without a new write control pulse.

Writing may be suspended. This can be done by holding the write control pulse at V_{IL} or less until the new page data is made available for

Memory Technology: 64k EEPROM

loading into the device.

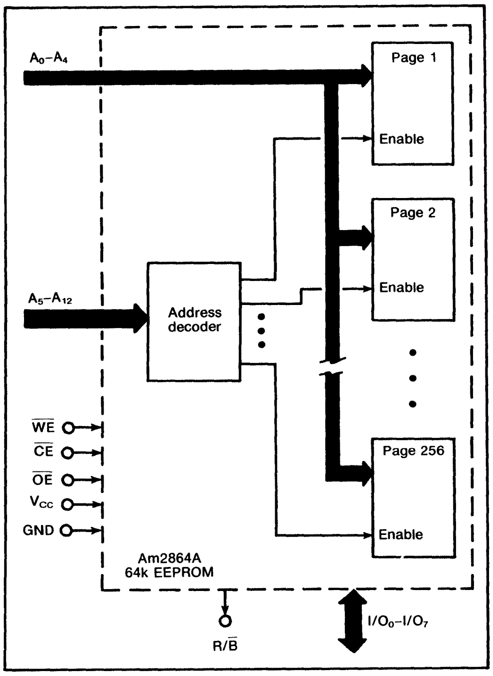
The byte address within a page is determined by the Y address (A_0-A_4), and the page address by the X address (A_5-A_{12}) (Figs. 2 and 3). The bytes within a page may be addressed in any order.

As discussed earlier, if it is inconvenient to use the Ready/Busy signal for interrupt modes, the data polling feature can be used to signal the completion of a byte or a page write cycle (Fig. 4). During a write cycle, inverted data from the last byte written will appear on the data outputs whenever a read cycle is attempted.

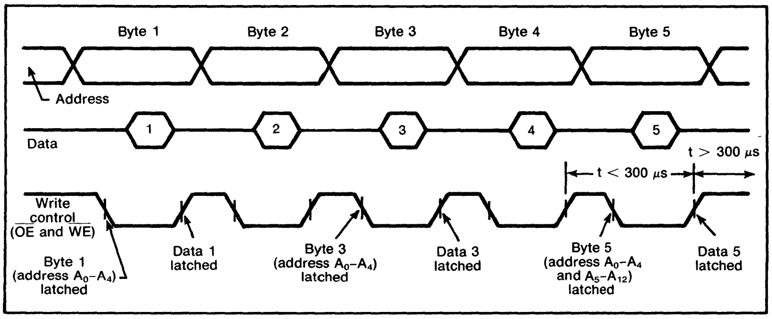
Make it teachable

The 2864A can be a major building block in robots that can learn new or revised tasks. Most industrial robots are programmed to perform simple functions. With high-capacity EEPROMs designed in, these robots will become "teachable"; that is, with assistance from an operator, the robot can reprogram itself.

Currently, most robots are heavy-duty units connected to mainframe computers and are used for spray painting or welding. The real demand for robotics, however, will be to assemble small or complex items. These robots must be smart and easily "teachable." That means that local processors and memory are a must. Hav-



2. When the 2864A is in the page mode, address A_{15}-A_{12} establishes which page will be accessed and address A_0-A_4 defines the byte on that page.


Byte 1
Byte 2
Byte 3
Byte 4
Byte 5
Address
Data
1
2
3
4
5
t > 300 μs
t < 300 μs
Write control ($\overline{OE}$ and $\overline{WE}$)
Byte 1 (address A_0-A_4) latched
Data 1 latched
Byte 3 (address A_0-A_4) latched
Data 3 latched
Byte 5 (address A_0-A_4 and A_5-A_{12}) latched
Data 5 latched

3. Here, five bytes of data are being loaded into the EEPROM's internal latches. Note that the internal address (A_0-A_4) is latched during each high-to-low transition of write control, as long as the duration between write control pulses is less than 300 μs. Because there is random access within the page, the page address (A_5-A_{12}) is latched only during the final byte to be written, which is byte 5. The bytes can be addressed in any order.

ing the processor on board makes fast local control possible, and nonvolatile memory allows the robot to be moved.

A simple learning system for the robot can be built, thanks to the flexibility of EEPROMs, as well as to the particular features of the 2864A.

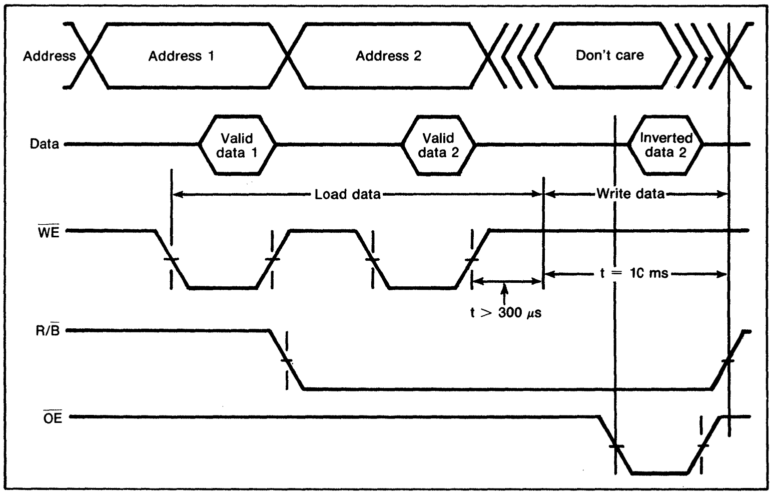
Consider a pick-and-place robot that can assimilate new information. Here, the robot's job is to pick up a plastic leaded chip carrier (PLCC); place a dot of glue on the bottom; and position it, within 4 mils, on the center of a pad on a densely packed printed circuit board. Because the part is to operate over the full military temperature range, the manufacturing process also requires an oven and a solder bath.

For the robot to do its job, the following design requirements must be satisfied:

- Nonvolatility and on-board reprogrammability, so that the robot can stand alone, be portable, and learn new information quickly and easily
- Fast loading of large amounts of data, because speed during storage cuts both the risk of losing data and the teaching time and allows the teacher or operator to modify action from a handheld controller or a personal computer
- The ability to operate in harsh environments (high temperature and solder and flux vapors)
- A data polling method of selecting interrupts, because software control is preferable to extra hardware
- High accuracy (to within ±2 mils), because of the packing density and expansion and contraction due to temperature

The last two concerns are the simplest to satisfy: A 16-bit industry-standard microprocessor will serve, because it is accurate enough to do the job (and canned software routines are available); also, servomotors are appropriate because they can be programmed much more accurately than stepper motors. As for the ability to operate in harsh environments, all the components must be specified for the military temperature range.

Nonvolatility is an important consideration because the unit needs to be portable—that is, power needs to be removable at any time—and cannot afford to lose the resident data. It also



4. The Ready/Busy output goes low when WE has been pulsed, data has been loaded into the latches, and 300 μs has passed since WE has switched from low to high. (CE is low for the entire diagram.) When OE goes low while R/B is low, the data at the outputs is the inverted data from the last byte written, indicating that a write operation is still under way.

must be adaptable—that is, the resident data must be updatable—and programmable on board. The last two concerns dictate a choice of three types of memory: hard disk, battery-backed RAM, or EEPROM. However, since portability is an issue, the hard disk is unsuitable, and the battery-backed RAM is inappropriate because of the extreme temperatures to be encountered. (Also, changing batteries is a nuisance for the operator.) That leaves EEPROM.

Because of the software functions required, a 64k device is necessary. As for fast storage, the page-writing mode of the 2864A fits the bill. Furthermore, it allows software-generated interrupt polling. (If the need exists, it also allows hardware-generated interrupts with its Ready/Busy pin.)

When designing with EEPROMs, the write cycle time always poses a problem, especially when there is a lot of data stacked up in the processing queue. The 2864A overcomes this problem with its page-writing mode, which allows up to 32 bytes of data to be written into the same page in a single 10-ms write time. The effective writing time is:

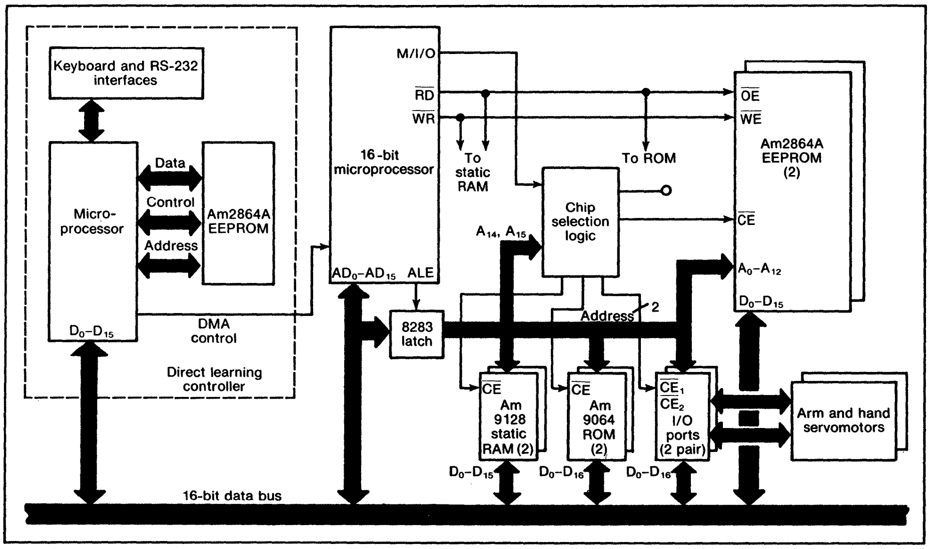
$$\text{total time} = \text{load time} + \text{write time}$$
$$\text{time/bit} = (\text{total time})/(\text{no. of bits})$$

For 32 bytes:

$$\text{total time} = 300\,\mu\text{s} + 10{,}000\,\mu\text{s} = 10{,}300\,\mu\text{s}$$

$$\text{time/bit} = \frac{10{,}300\,\mu\text{s}}{(8\text{ bits/byte})(32\text{ bytes})} = 40.25\,\mu\text{s/bit}$$

Since the design uses a 16-bit bus and two devices are operating in parallel (Fig. 5), the total



5. The 26864A is the key to enabling a portable pick-and-place robot to be reprogrammed quickly on the spot. Two, paired within the robot to make up a 16-bit data path, hold the coordinate and route information the robot needs to perform its task. A third, within the direct learning controller, permits an operator to "teach" the robot new information.

32-byte write time is 10,300 μs and the time per bit is:

$$\frac{10{,}300\,\mu\text{s}}{(2\,\text{devices})(32\,\text{bytes})(8\,\text{bits/byte})} = 20.13\,\mu\text{s/bit}$$

The final task is to design and implement an interactive teaching method. The direct learning controller, a subsystem of the robot, is the key piece of hardware for that.

The controller contains a processor, its own EEPROM, and a keyboard interface to allow the teacher/operator to communicate with the controller (see Fig. 5 again). The processor has two data buses, one internal and one external. The internal bus is connected only to the EEPROM, the keyboard, and the RS-232 interface. The external bus is connected to the system data bus.

Also, externally, there is a DMA control line that is connected to a polled interrupt in the main processor. This line is used during the teaching mode to shut off the main processor.

The learning controller is designed so that it can be conveniently removed from one robot and attached to another. This feature saves money because only one controller is then needed for any number of robots. Because of the controller's size, its portability, and the need to change data, the 2864A is a natural choice for the memory.

First the robot must be taught the function required. The teacher/operator simply puts the robot in the teaching mode and physically moves the arm to each point where the robot needs to perform a task. By pressing the position button, he or she loads the system RAM with the starting position, the rates of acceleration and deceleration through a prescribed arc, and the ending position. This occurs during regular operation; but when it occurs in the teaching mode, the RAM stores the data directly in the 2864As using direct memory access. During this time, instructions from the system processor are shut off.

The robot has generic instructions that the position coordinates and rate information can be plugged into. For example, a generic instruction located in the ROM instruction might consist of "Move arm from coordinates X Y Z to coordinates X′ Y′ Z′ with acceleration/deceleration vector D."

The table for this instruction would be filled up while in the teaching mode with position coordinates and rate information (Table 2).

Table 2. Vector table for the pick-and-place robot

[illegible]	Vector			
[illegible]	X	Y	Z	D
Rest position	0	0	0	0
Board assembly	5	9	7	3
Parts table	3	A	7	8
Glue pot	9	1	3	1

There are four vectors that direct the movement, each containing 16 bits of information. The EEPROMs are paired up to make a 16-bit data path. The 2864A pair can handle up to 32 of these vectors (or 32 bytes each) before a 10-ms storage cycle is required. The storage cycle can be performed during the time the teacher/operator is moving the arm assembly from location to location.

Now that the teacher/operator has moved the arm through the required motions and the information has been stored in the EEPROMs, the robot can begin to work. It will operate under the teacher/operator-given conditions until it is forced back into the teaching mode and given new instructions.

The closed-loop feedback path from the servo motors allows them to be very accurate. Because the EEPROMs store all position and velocity information, the robot can be manipulated by mechanical wear data that is also held in the memory. Thus, no matter how worn the equipment gets, the EEPROMs adjust for it.□

DRAM controller helps scrub errors

Scrubbing raises a DRAM's data integrity without adding complex timing and control logic

Jimmy Madewell
Memory Support Section Manager
Advanced Micro Devices Inc.
Sunnyvale, CA

Memory errors are generally classified in two ways: hard—a permanent fault caused by a structural defect like a metal short or open or a cracked die; and soft—usually a random single-bit failure in which data has been inverted from the originally stored data. Correcting the hard errors requires replacing the defective part; the soft ones must be handled as they occur by the memory-management system.

If a single soft error is not corrected soon after it occurs, a second error in the same data word would in time cause a double-bit error, which is far more difficult to correct. Large memories often contain sections that are not accessed on a regular basis. Thus the time between consecutive accesses can be longer than twice the system's meantime between soft errors, making the probability of double-bit errors high.

However, when a memory uses both dynamic RAMs (DRAMs) and an error-detection-and-corrrection (EDC) unit, those difficult-to-correct double-bit errors can be avoided by "scrubbing" the memory periodically to remove the single-bit errors.

Scrubbing is a background routine that combines with the refresh operation required for DRAMs to read one word from memory during each refresh cycle, check the word for errors, and, if an error is found, correct it and write it back into memory. Because each word in memory is accessed and checked for errors as the refresh cycle continues, double-bit errors are reduced to nearly zero, regardless of the possible low access frequency of any part of the memory. Scrubbing, therefore, adds high data integrity to a system without using complex timing and control logic.

A high-performance system for controlling memory using DRAMs can be segregated into three functional parts—DRAM controller, refresh and EDC timing control, and the EDC unit (see *Fig. 1*). A DRAM controller, such as AMD's new AM2968, manages the address path from the CPU to the memory. This device multiplexes row/column addresses, latches the addresses, drives the high-capacitance memory address and clock inputs, decodes addresses for selecting memory banks, maintains the correct refresh address, and assists in memory scrubbing.

Refresh and EDC timing control

A refresh and EDC timing controller is divided into seven functional parts—CPU status decoder, configuration and timing controller, arbiter, internal refresh timer, DRAM controller, EDC controller, and error and interrupt controller (see *Fig. 2*). The CPU status decode logic latches some of the CPU status and decodes the control bits to define the type of memory cycle. The refresh timer determines when a refresh cycle is needed, and an arbiter is used to resolve conflicts between refresh and read/write cycles.

The DRAM controller block receives decoded control signals from the CPU status decoder and timing signals from the configuration and timing control. This block then generates the interface control signals to be sent to the DRAM controller. The error and interrupt control logic receives the error-detected indication from the EDC circuitry. When an error is detected, the timing of the signals out of the controller is changed to allow the extra time needed to correct the bit in error. When multiple errors are detected, the CPU must be alerted through an interrupt.

A major consideration when designing a memory system is speed. If the memory is not faster than a specified threshold, the CPU will have to add an integral number of wait states after requesting a memory access. These wait states are multiples of the CPU clock period and can greatly increase the overall memory access time.

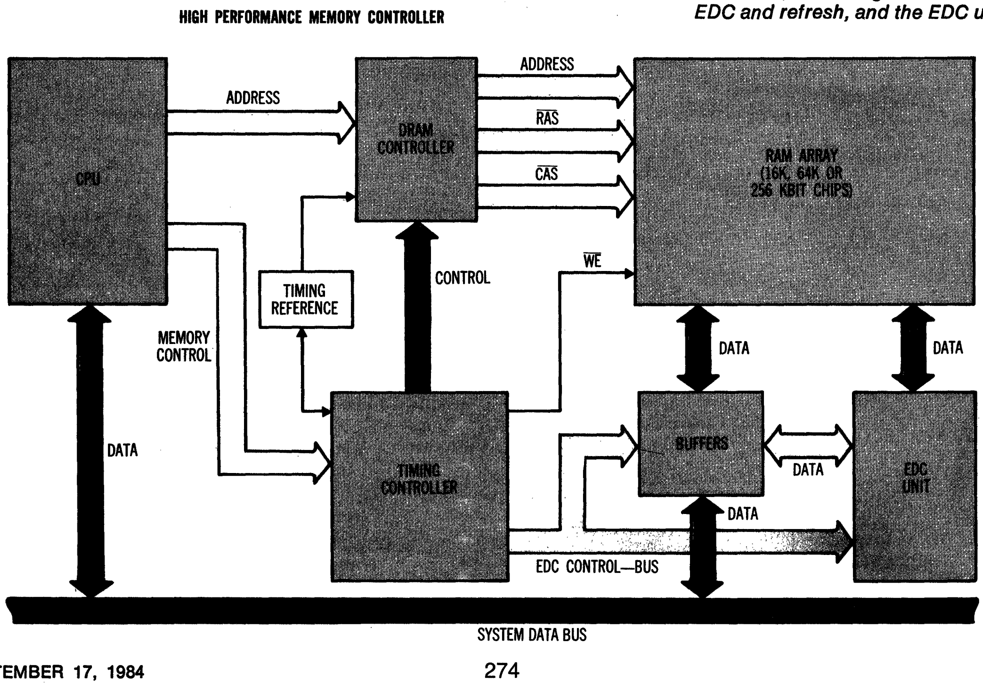
Error detection and correction

It must be remembered that if EDC is used, the memory-access time is increased, possibly putting the CPU into a wait state for one or more clock cycles. When using an EDC circuit, faster RAMs must be used for the same overall speed.

As larger and larger memories are used with microprocessors, minicomputers, and computer systems, data integrity becomes increasingly important. And as the number of bits contained in each DRAM increases, the soft-error rate caused by alpha particles also increases significantly. Memory designs using very high-density DRAMs (and perhaps even larger static RAMs) may require EDC to improve memory system reliability. Current studies show that reliability can be increased by 60 times or more if a suitable EDC scheme is employed.

There are two tradeoffs when designing a memory system that uses an EDC unit. First, the memory word width must be increased to include the necessary check bits. Second, extra time is needed to generate check

Fig. 1. A system for controlling a large dynamic RAM can be partitioned into three parts—the DRAM controller, the timing controller for EDC and refresh, and the EDC unit.



bits on write cycles and for generating and comparing check bits on read cycles. An EDC unit, such as AMD's AM2960, contains all the logic necessary to generate check bits for a data word according to a modified Hamming code and to correct any single error in the data word when check bits are supplied (see *Fig. 3*).

Operating on the data read from memory, the EDC unit corrects any single-bit error and detects all double- and some triple-bit errors. To accomplish this for 16-bit words, six check bits are needed (which is 38% of overhead); for 32-bit words, seven check bits are needed (22%); and for 64-bit words, eight bits must be used (13%).

When memory systems use EDC, the memory must be initialized following powerup to prevent errors from occurring during a read. The refresh and timing control logic, the EDC unit, and the DRAM controller work in concert to initialize the memory. The data input bits are forced to all zeros, while check bits are generated for this all-zero data. The refresh and timing logic then generates write cycles during which the all-zero data word with the cor-

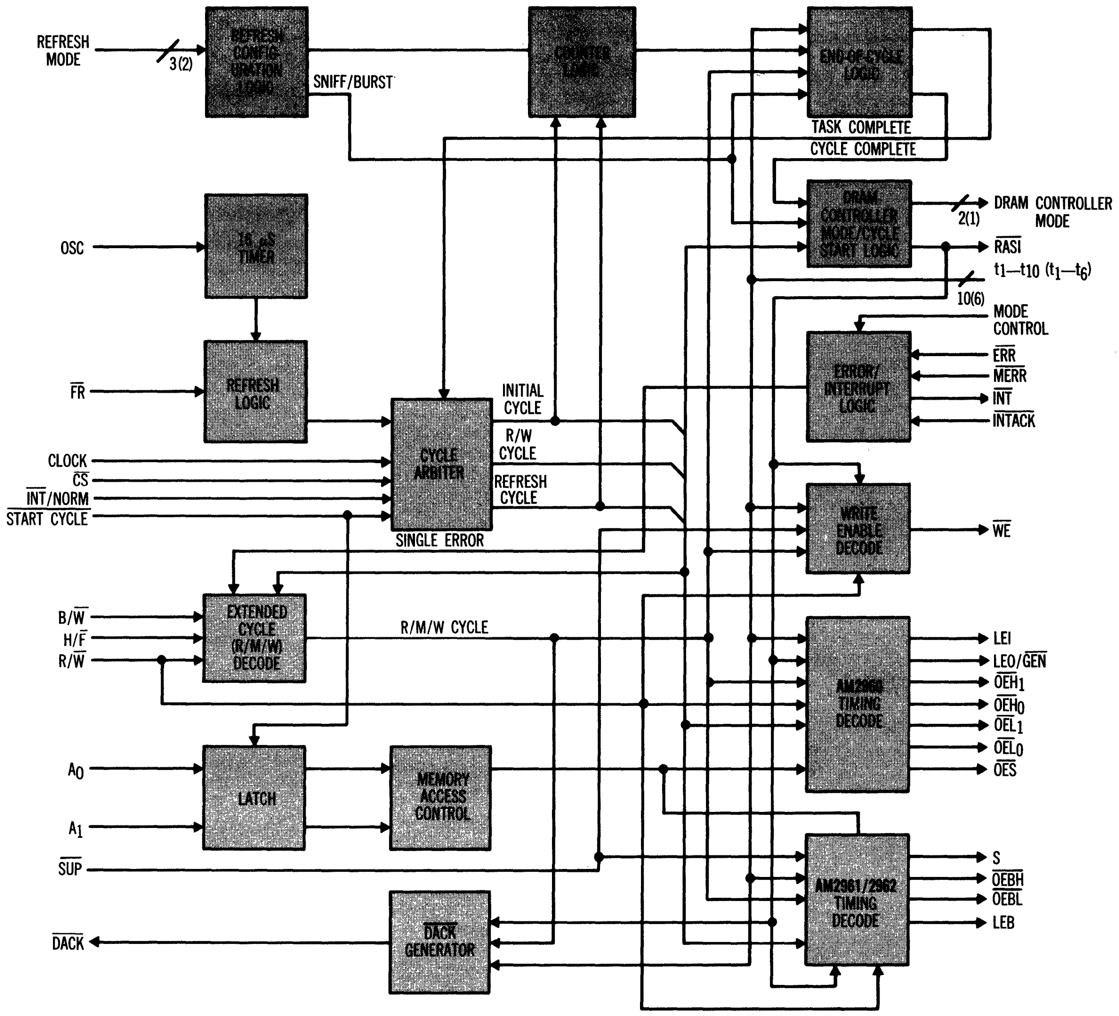


Fig. 2. Seven functional parts comprise the refresh and EDC timing controller—the CPU status decoder, arbiter, internal refresh timer, and controllers for DRAM, EDC, error-and-interrupt functions, and configuration and timing functions.

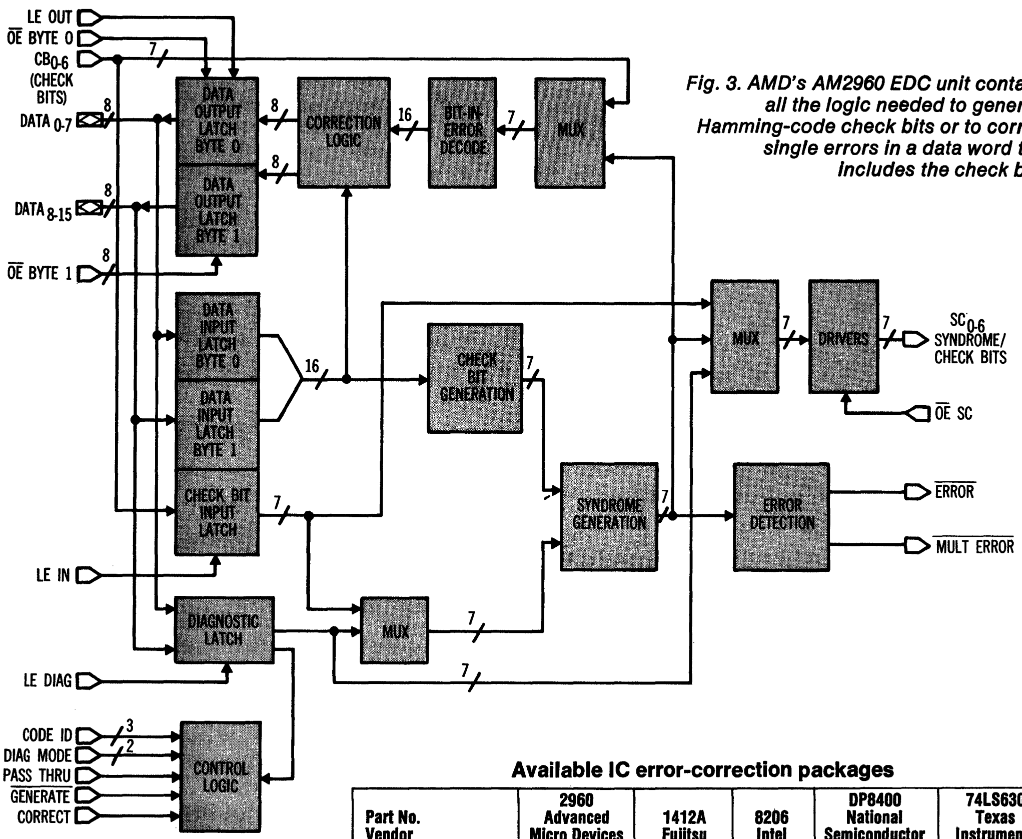


Fig. 3. AMD's AM2960 EDC unit contains all the logic needed to generate Hamming-code check bits or to correct single errors in a data word that includes the check bits.

responding check bits is written into all memory locations.

This initialization procedure is essential, since, after powerup, the memory will contain random data. A byte write attempted prior to initialization will cause the EDC unit to generate a multiple-error indication that stops the CPU. The memory must be put in a known state with appropriate check bits computed and stored for each location.

Memory and its support

Assuming a memory is constructed using typical DRAMs that require refreshing and row/column address multiplexing, accessing a DRAM requires two strobes to load an address. These signals are called row address strobe ($\overline{\text{RAS}}$) and column address strobe ($\overline{\text{CAS}}$).

First, the row address is strobed

Available IC error-correction packages

Part No. Vendor	2960 Advanced Micro Devices	1412A Fujitsu	8206 Intel	DP8400 National Semiconductor	74LS630 Texas Instruments
Packaging	48-pin CERDIP	64-pin Square-pak	68-pin LCC	48-pin CERDIP	28-pin CERDIP
Current Drain	400 mA	300 mA	250 mA	320 mA (typ)	230 mA
Word Width	16 Bits	8 Bits	16 Bits	16 Bits	16 Bits
Diagnostic Mode	Yes	No	Yes	Yes	No
Auto-Initialization	Yes	No	Yes	Yes	No
Expandable	Yes	Yes	Yes	Yes	No
Byte-Write capability	Yes	Yes	Yes	Yes	No
Second Source	Signetics Motorola		Western Digital		
Input to Error Detection (16-bit)	32 ns	39 ns	52 ns	30-40 ns	30 ns (25°C)
Input to Error Correction (16-bit)	65 ns	53 ns	65 ns	80 ns	45 ns (25°C)
Input to Check Bit Generation (16-bit)	32 ns	45 ns	58 ns	35-50 ns	65 ns (25°C)
Input to Error Detection (32-bit)	61 ns	62 ns	94 ns	75-105 ns	n/a
Input to Error Correction (32-bit)	105 ns	71 ns	145 ns	90-150 ns	n/a
Input to Check Bit Generation (32-bit)	60 ns	63 ns	96 ns	70-90 ns	n/a
Input to Error Detection (64-bit)	52 ns	98 ns	114 ns	145-185 ns	n/a
Input to Error Correction (64-bit)	102 ns	107 ns	165 ns	155-230 ns	n/a
Input to Check Bit Generation (64-bit)	42 ns	99 ns	116 ns	140-170 ns	n/a

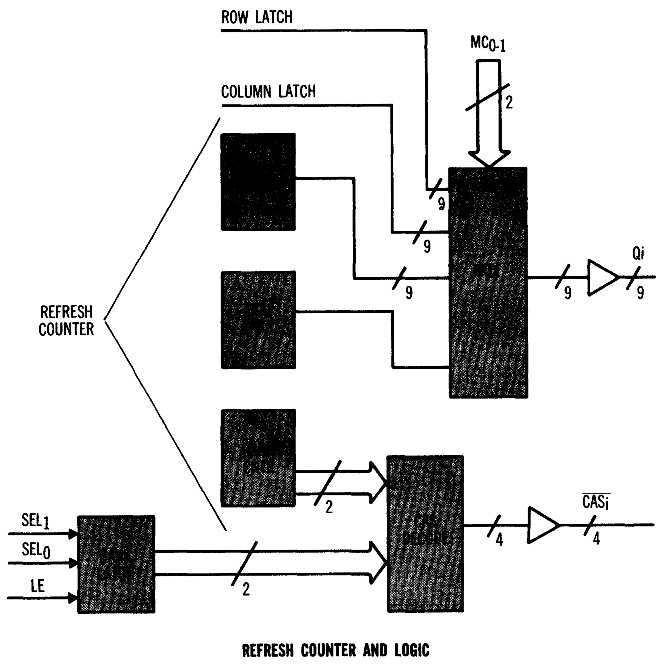


Fig. 4. Two functions—the column and bank counters and the CAS decoder —must be added to the basic memory control blocks to support scrubbing during refresh.

into the RAM by $\overline{\text{RAS}}$. Then the row address is removed, and the column address is set up and strobed into the RAM by $\overline{\text{CAS}}$. This sequence is needed to access any location in the DRAM. The $\overline{\text{WE}}$ signal is used to determine if a read or write is done.

Data is stored in a DRAM in the form of a capacitive charge, which decays over time unless the cell row is periodically accessed to recharge the cell and retain the data. Most systems do not access each row of every RAM within the cell decay time, so the RAMs must be refreshed. A $\overline{\text{RAS}}$-only cycle can be used to refresh all rows of the RAM.

Most dynamic RAMs have either 128 or 256 rows. In 128-row RAMs, all rows must be refreshed in 2 ms. In 256-row RAMs, all rows must be refreshed in 4 ms. Refreshing may be done in a burst, or normal refresh and access cycles may be interleaved. In the burst mode, all accesses are concentrated into one refresh period, whereas in the interleaved mode, a refresh occurs at a fixed rate (usually once every 16 μs) so that all rows are accessed in the refresh period.

Local refresh control gives rise to possible synchronization problems related to CPU-generated, read/write cycle requests. An arbitration circuit must be used to resolve conflicts. The arbiter circuit latches the input signals on the first memory request, inserts a timing delay during which possible conflicts are resolved, and grants a memory cycle.

DRAMs have several operating modes. The most common are read/write, $\overline{\text{RAS}}$-only, read-modify-write, page mode, and nibble mode. Any cycle that uses an active $\overline{\text{RAS}}$ signal can be used to refresh the addressed row of cells, whether or not $\overline{\text{CAS}}$ is also active. This feature is the basis of scrubbing.

Adding the final pieces

To obtain the scrubbing and refreshing operations using AMD' AM2960 EDC unit, the basic building blocks of a memory system are still needed, including a DRAM controller, memory, and EDC timing controller. Also, to support the scrubbing during refresh, two functions must be added—column and bank counters to the refresh row counter, and a $\overline{\text{CAS}}$ decoder (see *Fig. 4*).

During scrubbing, four $\overline{\text{RAS}}$ signals are generated—one for each bank of memory—together with one of four $\overline{\text{CAS}}$ signals. The $\overline{\text{CAS}}$ decoder decodes the two bank bits to select one of the four $\overline{\text{CAS}}$ output signals. The generated $\overline{\text{CAS}}$ output selects one of the banks of memory from which one word is read out. The word selected is determined by the row and column counters. At the end of each refresh cycle the row, column, and bank counters are incremented for the next refresh cycle.

Refreshed and scrubbed

As refreshing progresses, all $\overline{\text{RAS}}$ signals and one $\overline{\text{CAS}}$ at a time are generated, and all the cells of all banks are accessed for scrubbing. The refresh addresses are the least significant in the address field. For a memory array using 256K DRAMs grouped into four banks (one Mword of total memory) and a refresh period of 16 μs, the time needed to scrub the entire memory is 10^6 words x 16 μs, or 16 s. If all locations in memory are checked for errors every 16 s and single-bit errors are corrected and written back into memory, the probability of double-bit errors becomes very small.

Changes must also be made to the refresh and EDC timing controller. A control bit is needed to tell the EDC timing controller that scrubbing during refresh is to be performed. During scrubbing, the refresh and EDC timing controller change the timing from normal refresh timing to a regular read cycle, except when an error occurs, in which case a read-modify-write cycle is required. Of course, the design of an error-correction system around any other vendor's IC will differ from the above example. The table accompanying *Fig. 3* compares the features and specifications of the five currently available error-correction chips. □

Resettable RAMs speed cache systems

A small but fast, well-designed cache-memory system can raise the overall throughput of a computer. Computer memory appears to be operating at high speed even though the bulk of data is held in relatively slow memory devices

Bruce Threewitt
Manager of Memory Product Planning
Advanced Micro Devices Inc.
Sunnyvale, CA

Cache-memory systems are commonly used whenever a significant mismatch in speed exists between two levels of hierarchical memory. Memory devices are organized in a hierarchy according to size, speed, and costs (see *Fig. 1*). A cache-memory system allows a computer-based system to operate at the apparent speed of faster, more expensive memory components, but at the effective cost of slower, less-expensive components. This feat is possible bceause related pieces of information, which are to be processed together, are arranged in memory, in sequence, at adjacent addresses.

The major performance index for memory devices is access time—the time required to read or write data from or to the memory. Of course, access-time requirements vary with

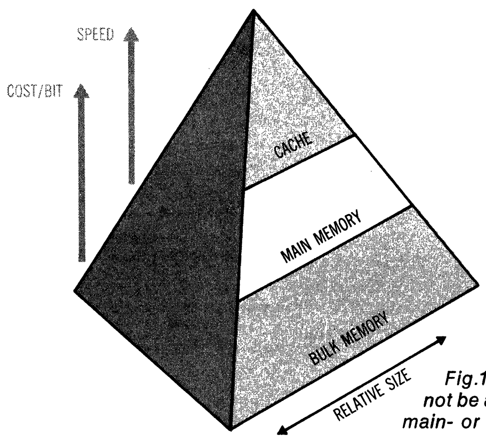


Fig.1. Fast RAMs used in a cache need not be as large as their counterparts in bulk main- or secondary-memory subsystems.

the application, and, generally, the faster the device, the more expensive it is on a per-bit basis (see *Fig. 2*).

The cache needs a very high-speed RAM. Current technology produces TTL-compatible RAMs with access times from 15 to 200 ns, depending on the cost and the power dissipation the application can tolerate.

The Advanced Micro Devices' Am-9150, for example, a 4-Kbit RAM with an access time in the 20-ns range, is considered a high-speed part suitable for a cache-memory system. Unlike most other fast RAMs, it is resettable, which gives the cache additional speed and simplifies the cache memory circuitry.

A typical cache-memory system (see *Fig. 3*) has two sections—the data cache, which stores the data used by the cache-memory system, and the address translator, or tag buffer, which stores the table of contents of the data cache. Both these systems must operate at the processor's clock rate—sometimes as fast as 20 MHz. Thus the memory components used must be comparably fast, with access times in the 50-ns range. Further, the number of words required is relatively small and the number of bits per word is relatively large. The memory chips' architecture, therefore, should reflect these simple requirements.

Is it in the cache?

When the processor asks for the contents of a particular memory location, the cache controller, in concert with the address translation buffer, determines whether the block containing the data is already in the data cache.

If the data cache contains the data, the address translation buffer and the cache controller enable the data cache, giving the processor access to the data at the high speed of the data cache. Finding the desired block of data in the cache is called making a hit, and the proportion of hits is called the hit ratio, a key measure of cache system performance.

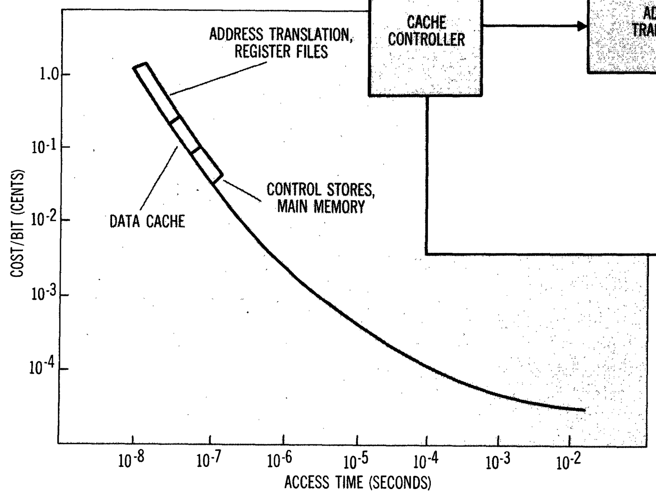
Conversely, if the desired block of data is not already in the cache, a miss occurs, and the cache controller must find the desired data in main or secondary memory and transfer it, along with the surrounding block, into the cache.

Such a transfer implies that an existing block of data in the cache must be written over (replaced). A replacement algorithm, whose effectiveness affects system performance, chooses the block to be replaced.

Whenever the processor writes new data into its cache memory, the cache controller implements one of two basic writing policies—a write-through or a write-back. With a write-through policy, the controller immediately writes a copy of the new cache data into main memory, so that main memory always contains current information. This approach incurs severe performance penalties if the number of write cycles demanded by normal system operation is large in relation to read cycles.

To avoid this penalty, the system can use a write-back policy, which

Fig. 2. (below) As might be expected, RAMs with the speed to serve in applications like cache memories and address translators cost more than those intended for a computer system's slower, bulk main-memory functions.



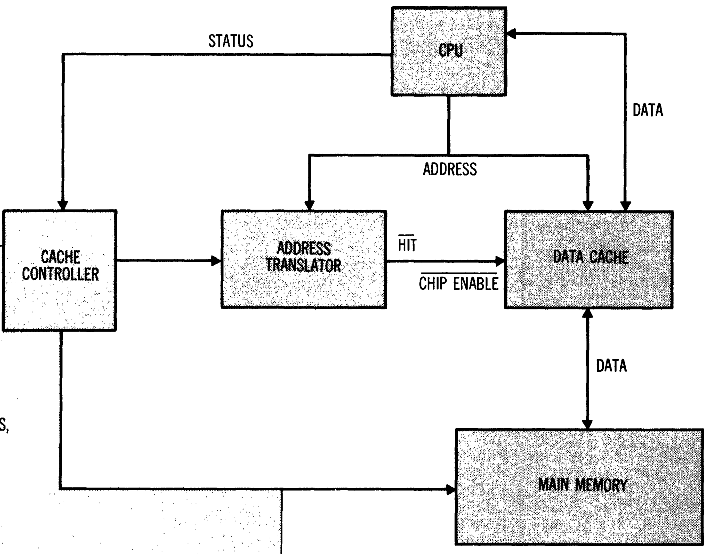


Fig. 3. Typically, a cache-memory subsystem (in green) sits between CPU and main memory. The higher the hit ratio, the lower the effective access time seen by the CPU.

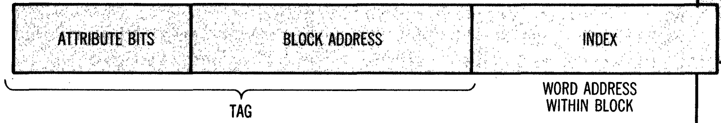


Fig. 4. Each block of memory in cache is identified by its address, bits that flag its contents' security status and existence/nonexistence on a disk, and an index to the words contained in that block.

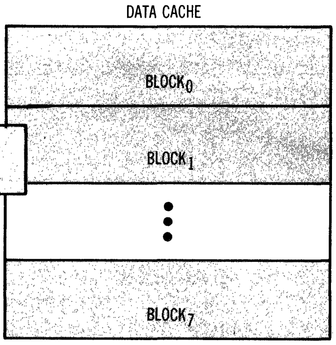


Fig. 5. Organizing data cache into sequential blocks of 128 to 1,024 words helps maximize the hit ratio to a value of almost one.

updates main memory only when it replaces a block. This approach requires the addition of a flag to each entry in the cache, which indicates whether data has been changed.

When a block containing "dirty" data (modified so that main memory and the data in the data cache no longer match), is replaced, the controller monitors the flag of each entry. If the flag is set, the controller writes that block into the main memory before discarding the data. Since hit ratios can be as high as 0.95, the penalty for the write-through is removed at the expense of slightly higher complexity and some additional memory.

Can contain attributes

Thus, the tag that labels each entry in the data cache may contain more than just the address of a particular block. The tag can also contain information about the attributes of a particular block as shown in *Fig. 4*. These attribute bits might include a valid bit, a dirty bit, security status, and so forth. In a multiuser environment, a change of context requires the clearing of the attribute bits of the previous cache contents.

Since the address translator is in the access path to the data cache, its component RAMs and comparators must be as fast as possible. And the need to clear attribute bits implies the need for an asynchronous memory-clear function. Context switches usually require a series of write operations to sequentially clear the tag RAM, at least in the attribute field.

Since the computer will next require information stored near the current memory location, the data cache is organized in blocks, (see *Fig. 5*). Block size, a key parameter in cache designs, may range from 128 to 1,024 words. Thus, the width of the index field in *Fig. 4* is $\log_2$ of the block size. The width of the block-address field reflects the number of blocks contained in the main memory. Whenever the computer asks for data not currently in the cache, the cache controller causes a block of main memory to be transferred into the cache. In a correctly designed cache memory, this transfer from main memory occurs infrequently, and the overall memory system speed is nearly equal to cache memory speed.

For optimum effectiveness, the performance characteristics of the data cache must match those of the CPU. Toward that end, data cache

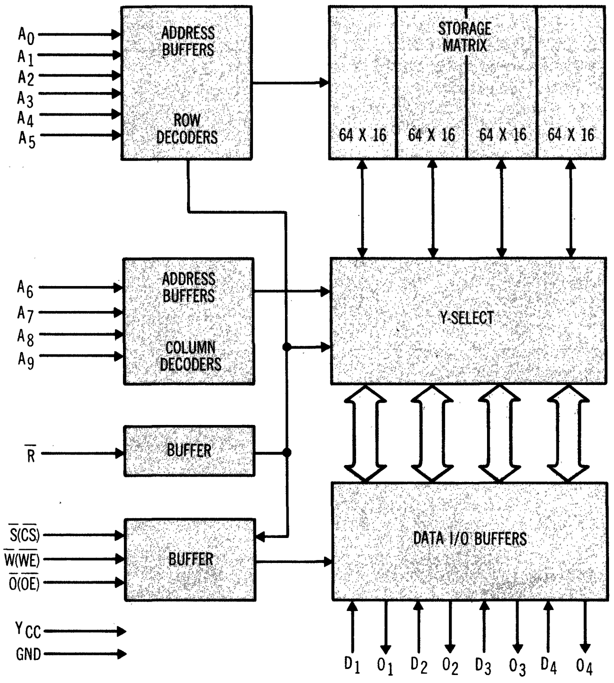


Fig. 6. Block diagram of Am9150 static RAM highlights chip's features: 1,024 x 4-bit organization, on-chip buffers, reset function, and separate input and output lines.

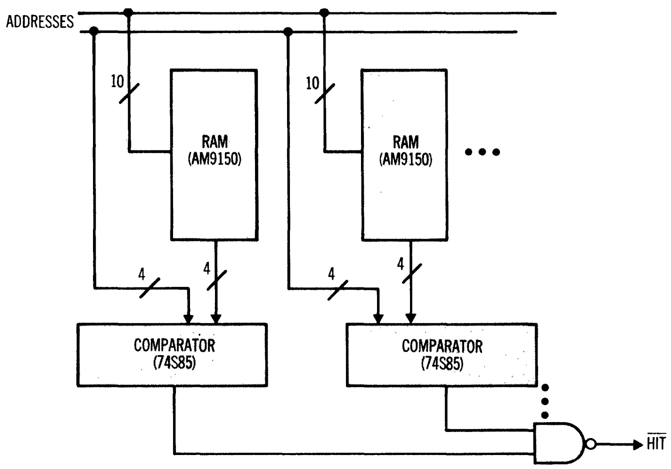


Fig. 7. The access time of a simple tag buffer is the sum of its constituents' speeds: 25 ns for the memory, 16.5 ns for the comparator, and 3 to 5 ns for the AND gate. Total speed is about 45 ns.

RAM components should be as fast as current technology permits. They should have separate inputs and outputs, and static RAM circuits to minimize cycle time.

Here, again, the organization of RAMs should reflect the shallow, wide nature of the data cache. Hence, a 4K RAM's preferred organization would be, say, 1,024 x 4 bits rather than 4,096 x 1. Package size and power dissipation make it impractical to include a large number of memory outputs in one RAM IC.

A cache-oriented static RAM

Figure 6 shows a static RAM designed for data-cache and tag-buffer applications. This device, the Am-9150, contains 1,024 words of four bits each, has separate inputs and outputs, and is packaged in a 24-pin slim DIP.

The Am9150, optimized for high performance at the system level, has separate data inputs and outputs plus 25-ns max access time. With the Am9150 and a fast comparator such as the 74S85, a simple tag buffer can be constructed (see *Fig. 7*).

The delay in the tag buffer is the sum of the Am9150's access time (25 ns), the comparator's delay time (16.5 ns), and the AND gate's delay time (3 to 5 ns). For TTL-compatible systems, a cache hit can be made in about 45 ns.

The Am9150 offers an asynchronous reset function, which clears the entire RAM in two cycles, or 50 ns, at most. This allows the attribute bits to be cleared without having to serially access all 1,024 tags, which saves a significant amount of time.

However, the reset operation can cause a great deal of noise in the system, due to the current spike caused by the simultaneous writing of all the bits in the RAM. Care should be taken to use bypass capacitors on all of the RAMs, so as to minimize this effect on the system.

The data cache operates by selecting the Am9150 with the composite hit signal from the tag buffer. Since the tag buffer delay is in parallel with the data cache, the system delay from tag input to data output is the sum of the tag-buffer delay and the chip-select delay of the data cache (15 ns, worst case), or about 60 ns, if a hit occurs. If the main memory has a cycle time of about 300 ns, and the hit ratio is 0.95, the effective memory cycle time is roughly t_{CYC} = (0.95) (60 ns) + (0.05) (300 ns) = 72 ns. The effective cache speed is about the same as that required by the CPU and costs about the same as the main memory. □

The Role of Programmable Logic in System Design

Om P. Agrawal and David A. Laws, Advanced Micro Devices, Inc., Sunnyvale, CA

In the race to produce faster, cheaper, more sophisticated systems in ever-smaller packages, designers are turning more and more toward application-specific ICs. Two major types of digital application-specific ICs in use today are gate arrays and programmable logic devices (PLDs). Gate arrays and PLDs have been commercially available for about 10 years, but are just now gaining popularity among designers after a long education and acceptance cycle. This article reviews the key features of PLDs, and tries to place them in perspective according to their value in a digital system design environment.

Programmable Logic Architecture

The familiar AND/OR structure of the programmable logic array (PLA) is at the heart of all current programmable logic devices (Figure 1). This structure can be used to implement a Boolean sum-of-products expression. Because every logical equation can be reduced to this form, the PLA configuration provides a versatile building block, which is now widely used in the design of VLSI circuits. When the AND and the OR matrices are both built with programmable fuse elements, the device is called a *field-programmable logic array* (FPLA).

Special cases of the FPLA provide for programmability of 1) just the AND matrix, with a fixed OR array, or 2) only the OR matrix, with a fixed AND array. These devices are known, respectively, as programmable array logic (PAL*) (Figure 2) and programmable read-only memory (PROM) (Figure 3) devices.

The first programmable elements offered on the market were PROMs. These devices were developed to help solve the problem of frequently changing microcode in microprogrammed machines. The key feature of PROM architecture is that the inputs are fully decoded by a fixed AND array. This array consists of a fixed address decoder and a programmable data-matrix memory array. A PROM with n input lines and m output lines contains an array of 2^n words. Each word is m bits wide. The input lines serve as an address to select one of 2^n words. When a pattern of n 0s and 1s is applied as an input, exactly one of 2^n decodes is output. Only one of the words in the memory array is selected, and the bit pattern stored in that location is transferred to the memory output lines.

The PROM configuration is useful for creating simple logic devices such as memory-address decoders. However, the fixed AND array limits its use in more complex applications. In such an array, multiple addresses cannot be used for the same word; therefore, the device can address only one word at a time.

*PAL is a registered trademark of and is used under license from Monolithic Memories, Inc.

Field-programmable versions of PLAs were the first products to be offered specifically for logic applications. Like a PROM, an FPLA with n inputs and m outputs can realize m functions of n variables. It can be visualized as having an address decoder (an AND array) and a data matrix (an OR array). Both the address decoder and the data matrix in an FPLA are programmable. Therefore, each AND gate can have "don't-care" inputs. Hence, multiple inputs or addresses can be used to select the same word. Because multiple AND outputs can be ON simultaneously, multiple words in the OR array can be selected at the same time, thus allowing a function to be divided among multiple FPLAs.

In the late 1970s, John Birkner and H.T. Chua of Monolithic Memories recognized that few system applications regularly required the FPLA to assign AND gates to specific output lines. This led to the development of the simpler PAL device (Programmable Array Logic) structure. Programmable array logic architecture is the mirror image of that of a PROM. It has a programmable AND array, but a fixed OR array. Thus, multiple addresses can select the same word, and multiple words in the array can be selected simultaneously. Like FPLAs, PAL devices overcome one of the key inefficiencies of PROMs by allowing as many inputs as needed for a particular implementation. However, because PALs do not have the fuses and programming and testing circuitry required by the OR array of an FPLA device, they are typically about 15% faster than FPLAs (for the same power consumption).

PLD Device Configuration

The three basic architectures for PLDs are augmented by additional logic capability for specific applications. These functions include registers, latches, feedback paths, and bidirectional I/Os. Many different PLD configurations have been developed that incorporate a variety of these features. These configurations include PROMs with registered outputs, FPLAs with buried state registers, and PAL devices with flexible I/O ports.

In this section, we will focus on programmable, bidirectional I/O pins. These let designers trade off outputs for inputs. Because this ratio can be programmed for a particular application, designers are not limited to a fixed number of input and output pins. The AmPAL22V10 (Figure 4) has 10 bidirectional pins and 12 input-only (dedicated) ports. Thus,

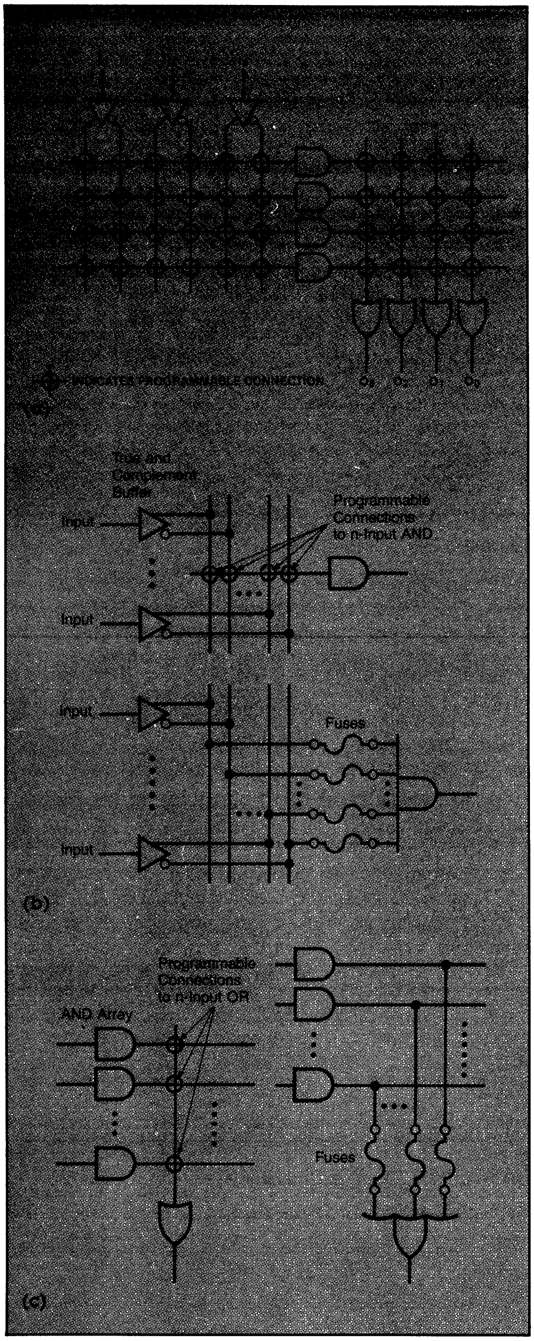

Indicates programmable connection
O3
O2
O1
O0
(a)
True and Complement Buffer
Input
Input
Programmable Connections to n-Input AND
Input
Input
Fuses
(b)
Programmable Connections to n-Input OR
AND Array
Fuses
(c)

FIGURE 1. FPLA structure (a), logic equivalent and conventional notation for the programmable AND array (b), and logic equivalent and conventional notation for the programmable OR array (c).

the device can support up to 22 inputs and 10 outputs, with the constraint that the total number of inputs and outputs cannot exceed 22. Earlier PAL devices typically supported up to 6 bidirectional pins, 10 dedicated inputs, and 2 dedicated outputs.

The bidirectional I/Os provide the ability to program the output buffer through a product term, and to configure the output as a dedicated output pin, as a dedicated input, or as a dynamically controllable input/output. Figure 5 shows the output for a 20-pin PAL device such as the 16L8. For an output configuration, the output buffer is always enabled (*i.e.*, all of the fuses associated with that product term are blown), and the pin behaves as an output pin. The logic function is fed back to the AND array. For an input, the product term associated with that output buffer is unused, thus configuring that pin as an input. Dynamically controllable input/output buffers (which occur when the output enable product term is enabled/disabled by a logical combination of one or more inputs) are useful in microprocessor bus-oriented applications, such as data steering, data storage, or data manipulation, or in connection with microprocessor handshaking protocols. This capability lets a pin be used as an input (for example, for left shifting) and as an output (for right shifting).

The degree of programmability and complexity of a PLD is determined by the number of fuses that form the programmable AND and OR gates. Each programmable AND gate is called a *product term*. The 20-pin medium-sized PAL devices (such as the 16L8 or the 16R8) have 64 product terms. The more recent 22V10 has a diode array of 5808 fuses, for a total of 132 product terms. Of these product terms, 120 provide logic capability and 12 are architectural or control terms that are used to program the variable-output macrocell configuration described below.

Each product term on the 22V10 consists of a 44-input AND gate. The outputs of these AND gates are connected to fixed OR gates. Product terms are allocated to OR gates in a variable distribution across the device, ranging from 8 to 16 wide. Thus, up to 16 logical terms can be evaluated in one output in a single clock cycle. The 64 product terms of the 20-pin PAL elements are allocated to OR gates 8 wide across the device.

In these PAL devices, the OR gates are metallized as either combinatorial (*e.g.*, in the 16L8, 16R8) or registered (*e.g.*, in the 16R8) functions at the output. In the registered versions, data is fed to the AND array internally from the Q output (Figure 5(b)). This feature is extremely useful for state-machine design. This 8-wide product term OR-gate, together with the register and feedback path, equals about 60 random gates in functionality when used to implement a state machine. If this value is extrapolated to the 8-, 10-, 12-, 14-, and 16-wide OR gates, the 22V10 becomes a 900-equivalent-gate device.

However, it is difficult to make a general rule for the gate-equivalent value of a PLD. The utility of the device depends on how well the logic can be accommodated by the AND-OR configuration. Because most design situations do not implement a device this efficiently (see box), the 22V10 can be considered a 600-to-700-gate general-purpose building block. Earlier PAL devices (such as the 16R8) have from 150 to 300 useful gates.

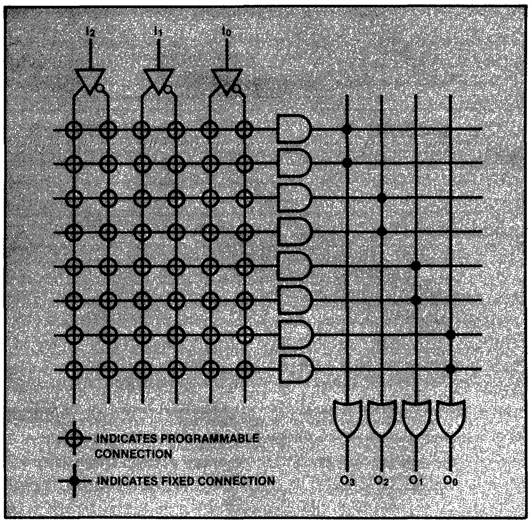


FIGURE 2. PAL structure.

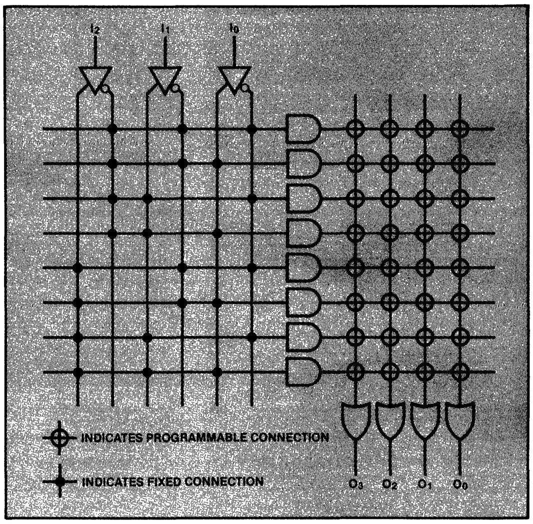


FIGURE 3. PROM structure.

A unique feature of the 22V10 is the output-logic macrocell that replaces a fixed combinatorial or sequential output configuration. Each macrocell can be programmed individually to operate in one of four modes: registered active HIGH or active LOW, or combinatorial HIGH or active LOW.

All PLD devices made today are fabricated with TTL bipolar technology. Junction-isolation processes are being replaced by oxide isolation in most new designs. Data sheets show worst-case propagation delays from input to output ranging from 25 ns to 35 ns. Typical delays for faster devices are approximately 12 ns. To duplicate this logic path with TTL SSI/MSI elements would require from four to six stages of gating. On this basis, equivalent typical delays (including the external buffer) are 2 ns to 3 ns per gate. Power dissipation is typically 300 mW for a 64-product-term PAL device, and 600 mW for a 130-product-term device. Using the gate analogy mentioned above, this dissipation is on the order of 0.5 to 1.5 mW per equivalent gate.

Logistical Considerations

Programmable logic elements are shipped from the factory in the blank form. The fuse pattern required to implement a specific logic function can be determined from the designer's equations using commercially available design software that runs on personal computers or workstations. This data is downloaded, via an RS-232 or similar link, to a PROM type programmer.

A simple logic design can be defined, entered into the system, and simulated, and prototypes can be programmed in less than an hour. The total equipment investment to do so can be less than $10,000. Because changes or corrections can be made on the spot and new devices can be programmed without delay, PLDs can support very interactive design techniques. Elements can go through many revisions. The complete system logic can be tested and modified several times until it works as desired. This is not feasible with gate arrays, because of the 8-to-10-week prototyping and rework cycles, and because of non-recurring engineering (NRE) charges ranging from $10K to $50K per iteration.

A second-order benefit of PLDs is that the devices can be quickly checked out *in circuit*. Although DC and AC timing-analysis programs for semicustom IC designs are improving rapidly, many characteristics, such as critical-path delays, switching noise, and line reflections, still cannot be predicted accurately before a chip is tested in the system.

An intelligent approach to system design combines the use of PLDs with custom ICs. The portions of a system that are well detailed and unlikely to change are incorporated into the largest custom or semicustom VLSI device that is practical. The logic that is likely to change is programmed into PLDs. This approach was taken by the designers of the DEC MicroVAX 1 (Louie *et al.* 1983).

Totally new system architectures are being developed entirely with PLDs. As designs are fixed, PLDs can be condensed into more complex semicustom devices. If production runs are low—up to a few thousand systems per year—it may be more economical to retain the PLD implementation because of the savings in non-recurring engineering (NRE) and tooling charges. These savings can be expressed graphically by superimposing the PLD price/volume curve on the popular family of curves (Figure 6) published in a recent issue of *VLSI DESIGN* (Beresford 1983).

System Design Considerations

Any digital system can be divided conceptually into three main blocks: the control path, the data path, and the interface. The control section interprets and executes macro instructions stored in main memory. It generates timing and sequencing signals, and provides decision-making signals. PLDs can be programmed directly as state machines, with up to 48 states, or can be used as building blocks for PROM-driven address sequencers.

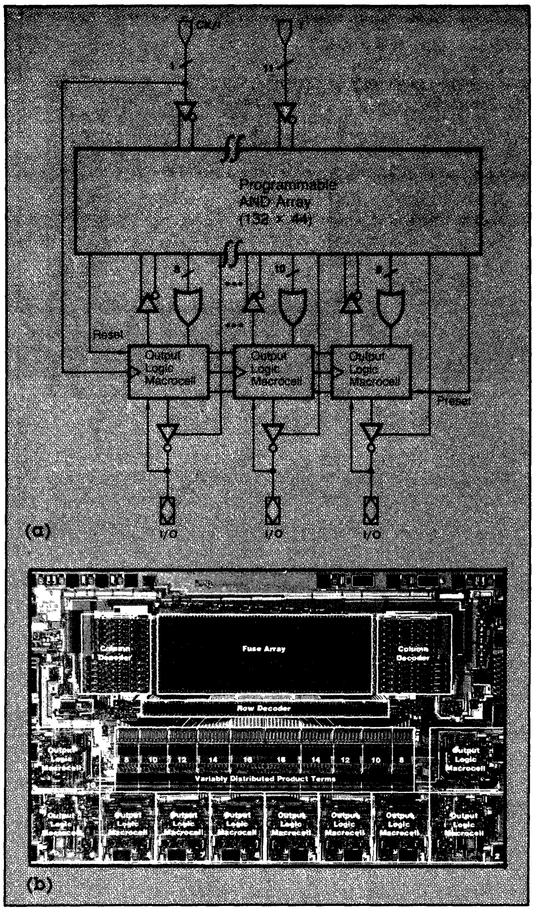


FIGURE 4. Block diagram (a) and partitioned microphotograph (b) of the AmPAL22V10 device.

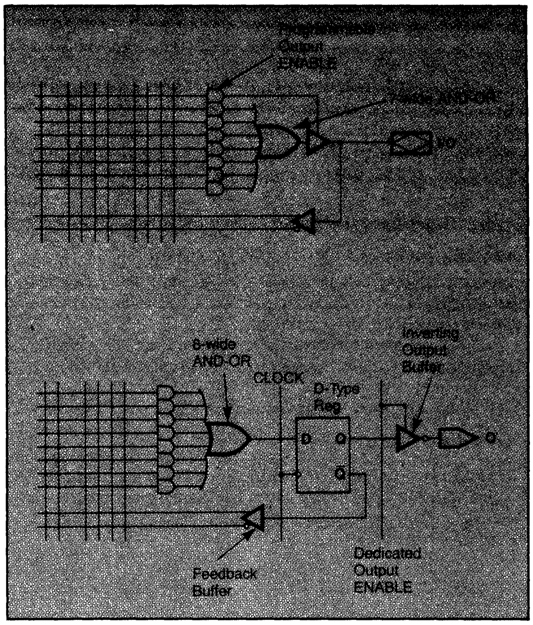


FIGURE 5. Output configurations for a PAL device: active HIGH bidirectional output (a) and registered output (b).

Data-path functions include manipulation (ALUs), storage (register files and pipeline registers), selection (multiplexing), steering, and shifting (*e.g.*, barrel rotators and shifters). The data path is usually the most structured portion of the system, and is likely to be defined early in the design cycle. Unlike the control section, the data path is the least likely to be changed. Performance and density are generally the most critical for data-path applications. For these reasons, standard VLSI devices or high-density/high-performance gate arrays are frequently used.

However, where volume is low, PLDs can be used for well-defined data-path functions, such as a barrel shifter (see box). In the example discussed in the box, the equations are suited to the architecture of PAL devices because output sharing of AND gates are not required.

Since their inception, PLDs have found widespread use in interface "glue" applications, where they are frequently used as state sequencers for controlling external logic for address multiplexing, timing generation, and refresh control in memory systems. For other interface applications (such as interfacing microprocessors to peripherals), timing changes can be made by adding or altering a term in the logic equation and reprogramming the device.

Retrofitting an existing, proven SSI/MSI design into PLDs involves identifying and combining as many separate packages as possible into single modules for replacement by a given PLD chip. This module can be completely combinatorial, or completely sequential; or it can be a combination of combinatorial and sequential logic. The retrofitting operation then proceeds with the following steps:

- Combining adjacent packages of 5 to 10 SSI/MSI functions into one module.
- Identifying input/output pin requirements of the overall block.
- Developing logic equations for the module; reducing input/output and product-term requirements with a Karnaugh map or by using other logic-minimization techniques.
- Once all inputs, outputs, and product terms are in their minimum form, matching these requirements to available PLD devices.
- If the requirements do not fit into a single PLD, then the function is re-partitioned.
- If the requirements fit well, and if the PLD is not used efficiently (with regard to I/O, product terms, etc.), then more SSI/MSI chips are combined and the process is repeated.

To exploit fully the capability of PLD functions, a new design should proceed in a top-down hierarchical way. First,

Designing Random Logic with PAL Devices

The AND/OR structure of PAL devices lets them implement any logic function, expressed in the sum-of-products form, provided the required inputs, outputs, and product terms do not exceed their capability. This capability is illustrated by the implementation of an 8-bit registered barrel shifter (Figure A) using the AmPAL22V10.

Design Requirements

A barrel shifter takes data input and cyclically rotates it an arbitrary number of bit positions. This cyclic rotation implies that data rotated from the most significant end of the shifter is returned to the least significant end.

An 8-bit registered barrel shifter requires at least 8 data inputs, 8 registered data outputs, 3 control lines to specify shift distance, a clock input, a chip-enable input, and an output enable that controls the three-state buffer on the register output.

Design Approach

The first step in PAL design is to generate Boolean equations for this function. Table A specifies the function of a complete end-around 8-bit barrel shifter. The following symbols are used for all logic equations:

* = AND
+ = OR
/ = NOT (invert)

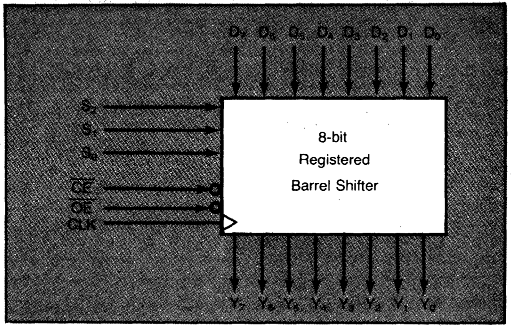
Figure B shows the Boolean equation for one of the eight outputs (/Y6) for the 8-bit registered barrel shifter. Seven additional expressions can be generated in the same way. These expressions indicate that a device with at least 76 product terms is required. Of the product terms, 64 are needed for the logic equations; the 12 other terms are needed to generate the output enable (8 for output configurations and 4 for input configurations).

The registered barrel shifter shifts 8 bits of data into the output registers as specified by 3 bits of control signals (S2, S1, and S0), synchronous with the clock input and provided the chip-enable pin is TRUE (/CE=LOW). When the chip-enable is FALSE (/CE=HIGH), the output register will be preset to all 1's. When the output-enable line (/OE) is LOW, the three state outputs are HIGH.

The next step is to select the particular PAL device to use for this function. (As noted above, this barrel shifter function requires a total of 14 inputs and 8 outputs.)

AmPAL22V10 can provide up to 22 inputs and 10 outputs—enough for 8 registered outputs, and enough for all other inputs (8 for data, 3 for control, 1 for chip enable, 1 for output enable, and one clock pin). The dynamic I/O programmable capability of the 22V10 lets this device meet this I/O requirement. The 76 product term requirements for this function are also met by the 22V10.

Note: As far as product-term utilization is concerned, 76 of the 132 product terms are actually being used. As far as the number of inputs/product terms is concerned, this functional implementation requires the use of 5 out of a maximum of 22 inputs.



D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
S_2
S_1
S_0
$\overline{CE}$
$\overline{OE}$
CLK
8-bit Registered Barrel Shifter
Y_7 Y_6 Y_5 Y_4 Y_3 Y_2 Y_1 Y_0

FIGURE A. Block diagram for an 8-bit registered barrel shifter.

```
/Y6 := /CE*/S2*/S1*/S0*/D6 +
       /CE*/S2*/S1*S0*/DS +
       /CE*/S2*S1*/S0*/D4 +
       /CE*/S2*S1*S0*/D3 +
       /CE*S2*/S1*/S0*/D2 +
       /CE*S2*/S1*S0*/D1 +
       /CE*S2*S1*/S0*/D0 +
       /CE*S2*S1*S0*/D7
```

FIGURE B. Boolean equation for one of the eight outputs for the barrel shifter.

Control Inputs			Input-to-Output Mapping							
S2	S1	S0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	D6	D5	D4	D3	D2	D1	D0	D7
0	1	0	D5	D4	D3	D2	D1	D0	D7	D6
0	1	1	D4	D3	D2	D1	D0	D7	D6	D5
1	0	0	D3	D2	D1	D0	D7	D6	D5	D4
1	0	1	D2	D1	D0	D7	D6	D5	D4	D3
1	1	0	D1	D0	D7	D6	D5	D4	D3	D2
1	1	1	D0	D7	D6	D5	D4	D3	D2	D1

TABLE A. Truth table for an 8-bit registered barrel shifter.

the designer should separate logical functions into control-path, data-path or interface-path functions. Then he or she should write the logical equations for those functions, minimize the input/output pins and product terms, and try to match the circuit to an appropriate PLD.

Software Design Aids

Once the designer has determined that the logic function can be implemented in a PLD, he or she must convert the equations into a pattern that can be translated by a programmer into the appropriate fuse location.

The first computer program to perform this was the PAL Assembler (PALASM*). Versions of this software have been written for many popular engineering machines, including IBM mainframes, the DEC VAX and PDP-11, and many

*PALASM is a trademark of and is used under license from Monolithic Memories, Inc.

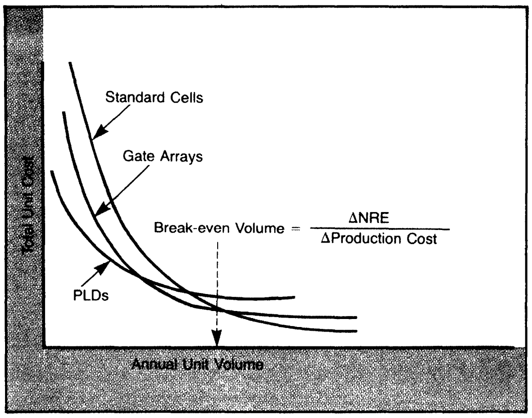


FIGURE 6. A popular family of curves indicates that the zero NRE charge for PLDs provides the lowest total unit cost at a low annual unit volume.

personal computers operating under CP/M and MS-DOS.

PALASM accepts an input file of Boolean equations and assembles the data into an output "fuse map" that is compatible with PLD programmers. It also lets designers input (in tabular form) test vectors to simulate and debug the equation input, as well as to test the finished device.

PALASM has also been incorporated, in EPROM form, into several PLD programmers, including those from Data I/O (Redmond, WA), Structured Design, Inc. (Santa Clara, CA), and Stag Micro Systems, Inc. (Sunnyvale, CA). The designer can assemble the equations directly into the programmer via a dumb terminal interface.

Several compiler-based high-level-language PLD design-aid software programs are now available or are being developed. The first commercially available product is CUPL, which was developed under UNIX and written in C by Assisted Technology of San Jose, CA. Data I/O is preparing a product it calls Advanced Boolean Expression Language (ABEL), and Advanced Micro Devices is beta-site testing its Programming Language for Programmable Logic (PLPL). These new generations of software will improve design productivity by allowing macro substitution and free-form comments.

Future Directions

New bipolar processes are expected to yield PAL devices having 15-ns worst-case propagation delays in 1984, and 10-ns or faster products by the end of the decade.

Several manufacturers are developing ECL devices that are expected to yield 5-ns worst-case delays. These products will be a complement to ECL gate arrays, which are finding increased use in large system design.

The same technological improvements that are increasing speed are also reducing power consumption in the 25-ns and 35-ns versions of the popular 20-pin PAL devices. These products can be expected to achieve 50% power reductions within the next two years.

CMOS technology is being investigated, both for reduced power in present configurations and to achieve structures providing equivalent complexities in excess of 1000 gates. However, CMOS programmable devices face challenging design problems because of the effect on performance caused by 30 or more inputs per gate. Work is also underway on CMOS non-volatile logic elements for both "one-shot" and in-system reprogrammability.

New trends in architecture will include greater use of programmable features. Outputs will be configured, either statically or dynamically, as registered, latched, or combinatorial functions, with active HIGH or active LOW polarity. Product-term distribution can vary dynamically among outputs, as needed. We can also expect features such as programmable product-enable terms for reset/preset on output registers and output enables. Diagnostic circuits will also aid device and system testing.

Current devices have relatively low complexity, because most vendors focus on small die sizes with limited I/O in order to fit devices into 20- and 24-pin slim packages which were designed for TTL SSI/MSI-level replacements. More complex bipolar PLDs with increased I/O capabilities, housed in 40-pin DIPs and 84-terminal leadless carriers, will be announced in 1984. These devices will offer up to 32 inputs/16 outputs and 64 inputs/32 outputs, respectively. The current state of the art in logic complexity (120 product terms) is predicted to expand by a factor of 10 to 20 by 1990. Although this complexity will exceed current gate-array densities, PLDs will always have some disadvantages in comparison with mask-personalized devices, because the programming and testing circuitry must be on board the PLDs. This overhead represents 10% to 20% of the active area of the die. □

References

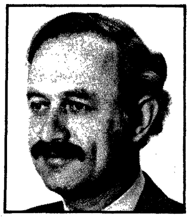
Beresford, R. December 1983. "Comparing Gate Arrays and Standard-Cell ICs," *VLSI DESIGN*.

Kitson, B., and J.B. Rosen. November 1983. "Logical Alternatives in Supermini Design," *Computer Design*.

Louie, G., T.Ho, and E. Cheng. December 1983. "The MicroVAX 1 Data-Path Chip," *VLSI DESIGN*.

About the Authors

Om Agrawal received the BSEE degree from Regional Engineering College (Rourkela, India), and the MSEE and Ph.D. in electrical engineering and computer science from Iowa State University. He is currently the manager of customizable logic in the Product Planning and Applications Department at AMD, where he is involved with the definition and development of next-generation semi-custom chips. Agrawal previously designed 16- and 32-bit minicomputer systems at Data General and at Rockwell.



David Laws joined AMD as director of marketing for the bipolar division, and currently is managing director for programmable logic products. His responsibilities include managing the business unit that is responsible for fuse-programmable logic and bipolar gate arrays. He received the BSEE degree in physics from Hull University (England) in 1962. Prior to joining AMD, Laws worked for Fairchild.

Super-minicomputer design using PALs

Bradford S. Kitson **and** ***B. Joshua Rosen*** **of Advanced Micro Devices, Sunnyvale, California discuss the recent trend towards using programmable logic devices in the latest minicomputers and look in detail at certain aspects of one particular super-minicomputer's design.**

Programmable array logic (PAL) devices have been a driving force behind the latest generation of 32-bit super-minicomputer designs. Most of these designs have used PALs for different cost/performance reasons. 'PAL-based' super-minicomputers include everything from redesigns of existing architectures for cost-reduction purposes or because of packaging limitations, to designs requiring rapid turnaround, to new high-performance architectures which have been specified with PALs in mind from the very beginning to take full advantage of these devices.

PALs allow the designer to architect, in a matter of minutes via fuse programming, a custom device or group of devices to implement a desired function precisely. This can be contrasted with fixed-function TTL SSI/MSI alternatives, which seldom seem to fit any application in the desired way. Because of the difficulty in force-fitting these fixed devices into custom applications, a designer using SSI/MSI usually pays the penalties of extra logic levels in the critical path and an increased package count. In some cases, fixed-function devices simply cannot be used to perform functions that PALs handle easily. PALs can also be compared to gate arrays, which allow the designer the capability to create custom devices via mask programming. A designer using a gate array must finalise his/her architecture early in the design cycle and cannot afford to make any errors. A mask change for a gate array can take months. PALs can be changed almost instantly allowing the architecture to be optimised or design bugs to be worked out without significantly affecting the design turnaround time.

Fig. 1

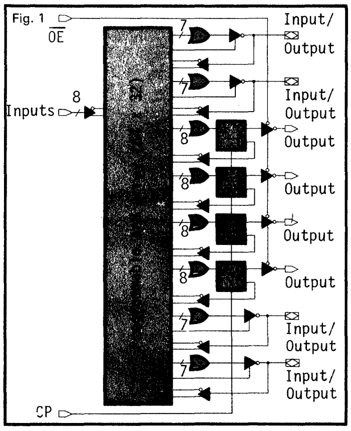


Fig. 2

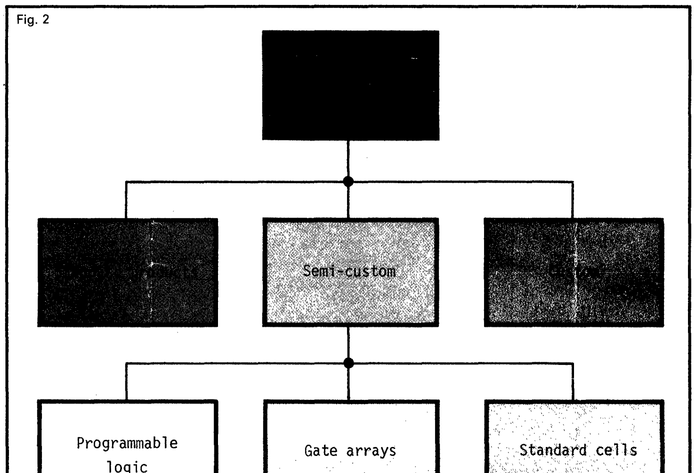


Figure 1 : The AmPAL16R4.
Figure 2 : Categories of digital logic.

Examples of PAL-based super-minicomputers include the VAX 11/730 from Digital, the MV8000 and MV10000 from Data General, and the Computervision Analytic Processing Unit (APU). The VAX 11/730 is a reimplementation of the original VAX 11/780, at one quarter the performance and one-tenth the board space. The MV8000 was the first machine to utilise PALs, chosen by its designers to allow Data General to create a machine in the shortest possible design cycle and 'catch up' with their competitors. The MV10000 is an upgrade in performance of the MV8000. The Computervision APU is a machine that was designed for high performance with PALS in mind.

PAL architecture. The Basic structure of a PAL is a fuse-programmable AND gate array that drives fixed connection OR gates, allowing logic to be implemented in sum-of-products (AND-OR) Boolean form. AND-OR is the most common and straightforward form of representation for digital functions — it is in fact the first representation taught in basic logic design courses at college. The AmPAL16R4 is shown in figure 1. The true and complement version of each of the sixteen inputs is connected via fuses to each of the sixty-four AND gates in the device. By selectively blowing the appropriate fuses, the PAL can implement any logic function as long as the number of inputs or AND gates required does not exceed the number provided in the device chosen. PALs provide additional features, such as programmable I/O pins and registered outputs with internal feedback, which further enhance their capability to implement logic functions efficiently, (figure 1). Programmable I/Os are especially useful for trading off the number of device outputs for inputs to fit the exact number required by the logic function(s) being implemented. Internal registered feedback is desirable when implementing complex state machine designs.

Design alternatives. PALs have become the technology of choice in super-minicomputer design, but are by no means the only choice. Figure 2 is a general table of logic design alternatives. Categories include standard products (previously referred to as fixed-function devices), programmable logic, gate arrays, and fully-custom logic devices. The main alternatives to PALs in super-minicomputer design are standard products and gate arrays.

The best choice for any given function depends upon the capabilities of the design alternatives, the constraints of the design, and the actual function being implemented.

Typically, the six basic constraints of any design are performance (speed), cost, density (packaging), power dissipation, reliability, and design turnaround. A prioritisation of these constraints will usually define the logic alternative a machine is based on. Actual implementation trade-offs are made at the function level. The three basic functional portions of a design are data path, control path, and interface.

Standard products are defined as devices created by the IC manufacturer for a wide market and are not alterable by the user. Examples of standard products are TTL SSI/MSI, fixed instruction set MOS microprocessors, and microprogrammable LSI building blocks. The main advantage of standard products is cost. These devices are usually multiple sourced and produced in high volume, resulting in lower individual device costs. In a design where cost is the main concern, and performance, power dissipation, density, and design turnaround are of little or no importance, standard products are probably the best choice.

In a design such as a super-minicomputer where these other considerations are of greater importance, the inherent disadvantages of standard products prevent designers from

Fig. 3

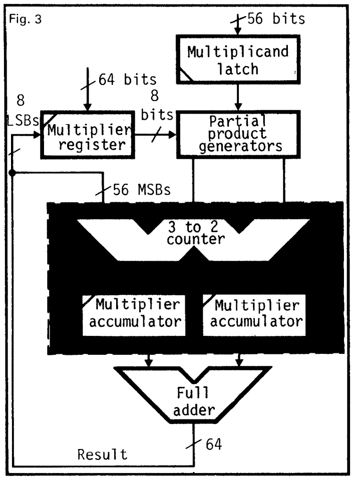


Figure 3: Computervision's APU multiplier.

Fig. 4

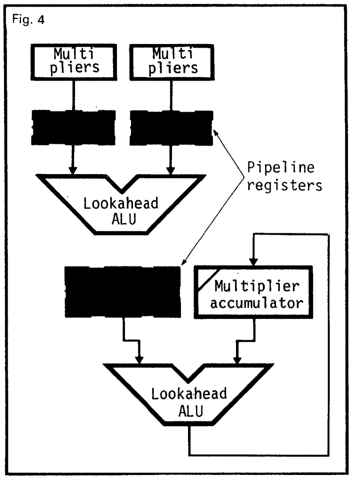
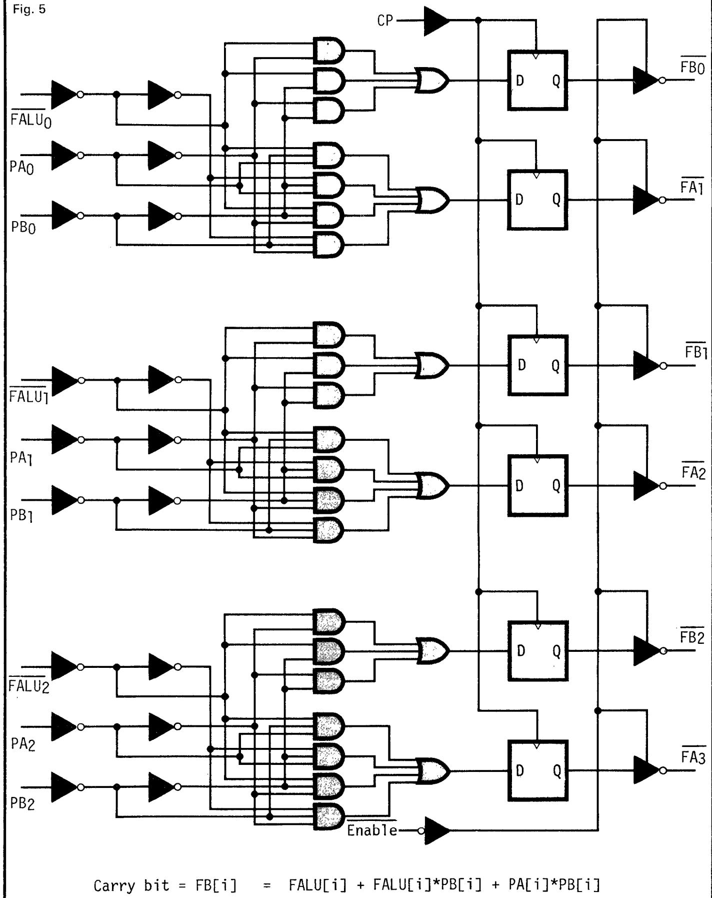


Figure 4: Two-level pipelined multiplier approach.

Fig. 5



```
Carry bit = FB[i]   =  FALU[i] + FALU[i]*PB[i] + PA[i]*PB[i]

Sum bit   = FA[i PLUS 1]  =  FALU[i] XOR PA[i] XOR PB[i]
```

Figure 5: AmPAL 16R6 3-to-2 counter

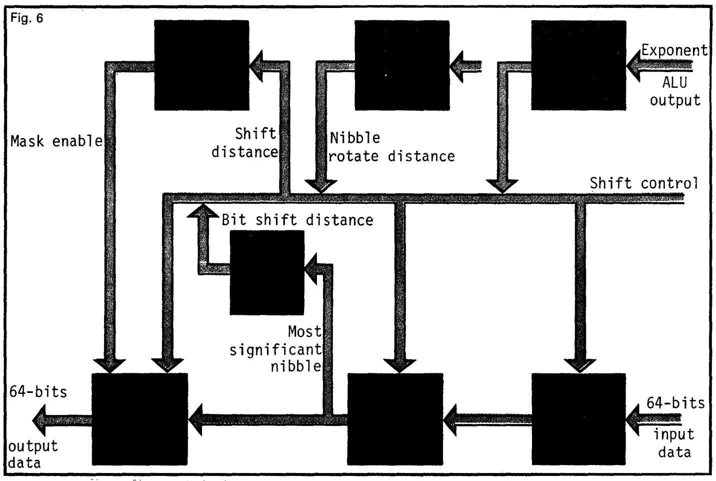
Fig. 6


Figure 6: 64-bit 'Rosen shifter'

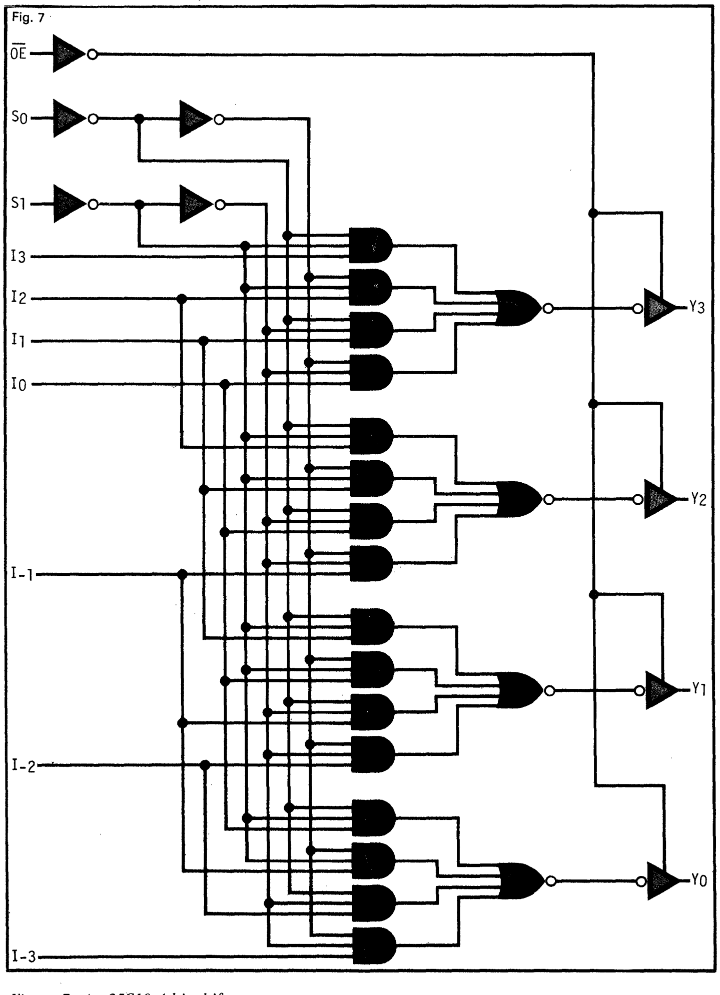
Fig. 7


Figure 7: Am25S10 4-bit shifter.

basing machines on them.

While SSI/MSI devices offer the fastest individual gates, on a system level they provide worse density, power dissipation, and reliability characteristics than any of the alternatives. Additionally, these devices offer poor design turnaround characteristics because changes usually require PCBs to be laid out again. In most super-minicomputers, the use of SSI/MSI gates is limited to selected critical path functions requiring one or two gate levels (at the expense of density and power dissipation, of course), and to interface applications, such as bus buffers, latches, registers and transceivers.

Standard LSI products, mostly bipolar, can offer exceptional performance, power dissipation, density, and reliability characteristics, but their inflexible architectures limit their range of applications. Superminicomputer designers rely on proprietary, highly complex architectures to differentiate their designs from their competitors'. LSI, by its very nature, imposes an architecture on the designer. Most applications are limited to general-purpose, well-defined functions in the data path, such as parallel multipliers and ALUs, because these functions do not impose on the architecture. Additionally, the critical timing requirements of data path functions can benefit from the high performance characteristics of LSI.

Gate arrays are semi-custom devices in that their internal interconnection is specified by the end user for a specific application. The main advantages of gate arrays are the ability to customise a design, and density. In a proprietary design where density is an overriding constraint, a gate array-based design may be the best choice.

The main disadvantages of gate arrays are cost and design turnaround. Gate arrays are basically the opposite of standard products: each custom device is single-sourced and low volume. Gate arrays are not cost-effective unless the application is density limited or the volume is very high. Gate arrays can adversely affect design turnaround in two ways. First, the system designer must do two designs, the IC and the system in which the IC is used. Second, should any change in the gate array be necessary, whether to optimise the architecture or work out a design bug, new masks must be created. Each mask iteration of a gate array can take months.

Most super-minicomputer designers try to minimize the turnaround disadvantages by limiting their use of gate arrays to portions of the design that can be defined early on ('cast in concrete'), and hedge their bets by placing logic around the gate array that may be able to cover for any design bug(s) discovered later on. As in LSI, what is cast in concrete is usually the data path, but most likely more of the data path is integrated onto the device because it can be optimised for the specific requirements of the design.

A proprietary sixteen-bit ALU slice might be a typical gate array. Gate arrays are also used to interface multiple on-board buses together as data-path 'glue'. Control path and interface applications tend to be too likely to change for gate arrays not to involve the risk of longer design-turnaround times. The control path tends to be writable-control-store (WCS) based to allow changes to occur even after the product has been introduced. (Many designers consider WCS to be RAM-based programmable logic!). Interface applications are often changed because this is where one designer's board must interface to another's.

Consider, for example, the critical timing between a cache, an instruction fetching unit and multiple register-files. The exception handling capability required for typical operations, such as a cache miss, can be an essentially indeterminant problem. Additionally, gate arrays do not have the drive capability required of interface applications.

Programmable logic devices are also 'semi-custom' in that they are created by the IC manufacturer but alterable in house by the end user, by fuse programming, for a specific application. PALs are programmable logic devices designed specifically for logic-oriented applications. The main advantages of PALs are the ability they give the designer to create custom devices and their fast turnaround time. They fall behind gate arrays and LSI only in density and power dissipation, and behind SSI/MSI in individual device costs (but note that PALs may be cheaper on a system basis).

Fig. 8

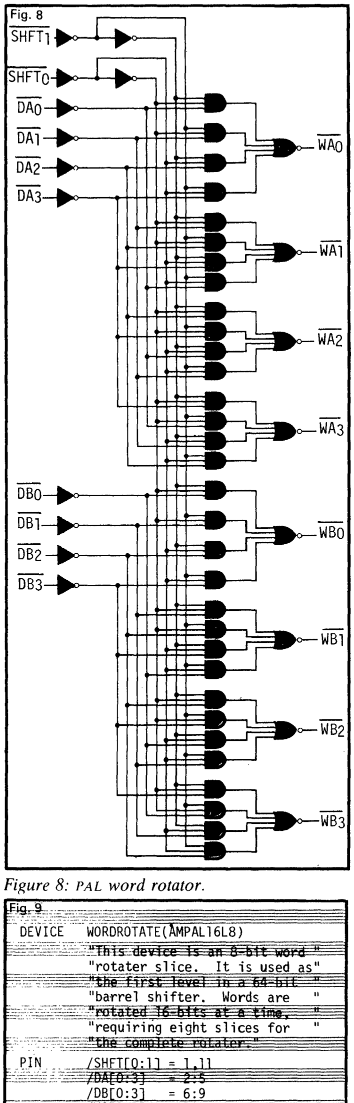


Figure 8: PAL word rotator.

Fig. 9

```
DEVICE    WORDROTATE(AMPAL16L8)
          "This device is an 8-bit word "
          "rotater slice.  It is used as"
          "the first level in a 64-bit  "
          "barrel shifter.  Words are   "
          "rotated 16-bits at a time,   "
          "requiring eight slices for   "
          "the complete rotater."
PIN       /SHFT[0:1] = 1,11
          /DA[0:3]   = 2:5
          /DB[0:3]   = 6:9
          /WA[0:3]   = 19:16
          /WB[0:3]   = 15:12;

BEGIN

CASE(SHFT[0:1]
          BEGIN
          0:        BEGIN
                    WA[0:3]  = DA[0:3];
                    WB[0:3]  = DB[0:3];
                    END;
          1:        BEGIN
                    WA[0:3]  = DA[1:3,0];
                    WB[0:3]  = DB[1:3,0];
                    END;
          2:        BEGIN
                    WA[0:3]  = DA[2:3,0:1];
                    WB[0:3]  = DB[2:3,0:1];
                    END;
          3:        BEGIN
                    WA[0:3]  = DA[3,0:2];
                    WB[0:3]  = DB[3,0:2];
                    END;
          END;
END.
```

Figure 9: PAL word rotator specification.

The favourable performance and density characteristics of PALs have allowed them to find significant applications in the data path of superminis. The typical super-minicomputer uses PALs along with LSI and/or gate arrays in the data path. The PALs 'glue' the LSI and the 'cast-in-concrete' gate arrays into the system.

The design turnaround capability of PALs can be used to optimise the data-path architecture and, in some cases, a bug in a gate array can be 'fixed' by reprogramming the PAL(s) around it. Typical PAL data-path functions include barrel shifters, masking, code conversion, and efficient interface of multiple buses.

The design turnaround capability of PALs becomes especially beneficial in the control path and interface portions of a design. Most control path functions are highly random and are prone to change or error. WCS is used to allow design changes to be made via re-writing microcode. Most designs use PALs extensively in the control path to allow easy changes to be made that cannot be done in microcode, or would adversely affect system performance. For instance, one super-minicomputer manufacturer only allows a gate array to be used in the control path if it takes more than eight to ten PALs to do the same function!

The interface portion of a design is perhaps where PALs are most valuable. From a density standpoint, PALs allow the interface portion of a design to be merged with the data path 'glue' function. From a design turnaround standpoint, the interface portion of a design is just as likely to change as the control path, but does not have the benefit of WCS. Should a designer of one board need to change the interface, it may affect the designs of many other boards. Without the ability to update all of the other boards easily, by re-programming a PAL or PALs, just keeping track of design changes can become unmanageable. Additionally, PALs can provide the drive capability required by this application area.

The following sections describe portions of the Computervision Analytic Processing Unit (APU), examining the trade-offs between PALs and other digital logic alternatives.

PALs are used in conjunction with standard products, such as ALUs, LSI multipliers and memory devices to create a unique, patented architecture that was not previously feasible in standard TTL SSI/MSI.

The Computervision APU was designed as a very high speed 32-bit super-minicomputer intended for engineering applications. PALs were used extensively in the design. Instead of just replacing TTL SSI/MSI, the designers of the Computervision machine used PALs as customisable logic building blocks. The result is a machine which is twice as fast as competitive designs without an increase in board space.

The APU processor board set is divided into four modules: The parser/sequencer, which contains an instruction processor that fetches and decodes instructions in parallel with the execution unit; the control processor, which performs address and integer computations, the floating point pipe, which performs both scalar and vector floating point operations; and the cache/address translation unit, which contains a 256-slot area page table entry cache and a 16K byte memory cache. Approximately 25% of the chips in the APU board set are PALs.

The APU is the first implementation of Computervision's new CPU architecture. As such, many aspects of the design were subject to change. The use of PALs permitted the designers of the machine to modify the hardware rapidly to fit the needs of this evolving architecture. In addition, new features and performance enhancements could easily be implemented with minimal impact on the development schedule.

The ability to generate a very large number of essentially custom ICs — over 200 different PAL codes are used in the APU — significantly reduces the processor's size while enhancing its overall performance. The net result is apparent when one considers that although the APU and Digital's VAX-11/750, a gate array based machine, consume exactly the same amount of board space, the APU is more than twice as fast.

In fact, the Fortran performance of the APU is substantially faster than that of the VAX-11/780, a machine which consumes 5.2 times as much board space as the APU.

The APU floating point pipe (FPP) was designed as a high speed arithmetic extension to the APU execution engine. Unlike other comparable machines, the floating point arithmetic unit of the APU is an integral part of the internal architecture and not an optional add on. As a result, the FPP is used not only to accelerate scalar and vector floating point arithmetic, but also to perform byte, word, double word and quad word string operations. In addition, the FPP is used to enhance the performance of important non-floating point instructions such as Procedure Call and Return.

The FPP board uses a total of 79 PALS for both control- and data-path applications. The remainder of this paper focuses on two sub units of the APU FPP: the multiplier and the barrel shifter.

The multiplier section is the heart of the FPP, see figure 3. Double precision floating point multiplication requires the calculation of a 56-bit × 56-bit product. Unfortunately, 56 × 56 parallel multipliers do not exist on silicon. The best cost/performance solution is to use a number of smaller multipliers to build an intermediate sized parallel multiplier and then produce a large product (56 × 56) in multiple cycles.

The partial product generator logic of the FPP employs seven Am25S558 8 × 8 multiplier slices to implement an 8× 56-bit multiplication array. Each multiplier chip produces a 16-bit product. In general, the most significant eight bits of each partial product generator must be added to the least significant eight bits of the next higher slice to generate the full 64-bit partial product. Exceptions are the most significant and least significant eight bits.

The technique also demands the ability to accumulate partial products with the partial products from previous cycles. Thus each cycle must be accompanied by two additions: the partial product summation and the intermediate product accumulation. The most straightforward way to accomplish this task is to follow the multipliers with two levels of look-ahead adders, usually 74S181s.

This technique results in a nanocycle time which is approximately three times longer than the partial product generation time of the 8×8 multipliers. This is clearly unacceptable. The scheme can be modified, however, by adding registers between each level of logic (figure 4). By pipelining the multiplier in this fashion, the

Fig. 10

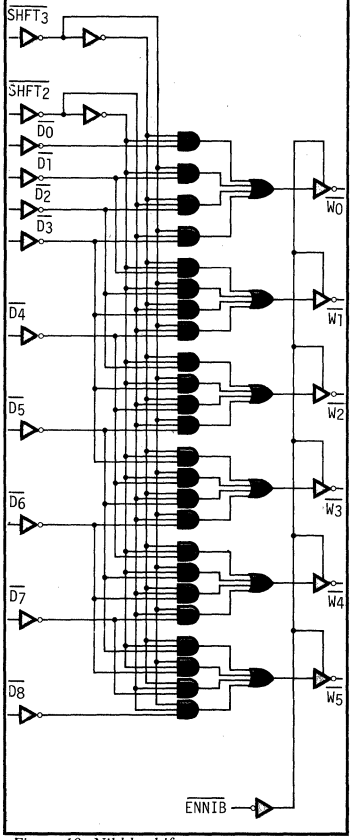


Figure 10: Nibble shifter.

Fig. 11

```
DEVICE    NIBSHIFTER(AMPAL16L8)

          "This device is a six-bit nibble  "
          "shifter slice.  It is used in the"
          "second level of the 64-bit barrel"
          "shifter.  Data may be shifted up "
          "0,4,8 or 12 places."


PIN       /SHFT[2:3]  =  1,18        /ENNIB = 11
          /D[0:8]     =  2:9,13
          /W[0:5]     =  19,17:14,12;


BEGIN


IF (/ENNIB) THEN ENABLE(/W[0:5];


CASE(SHFT[2:3])
          BEGIN
          0:      W[0:5]  =  D[0:5];
          1:      W[0:5]  =  D[1:6];
          2:      W[0:5]  =  D[2:7];
          3:      W[0:5]  =  D[3:8];
          END;

END.
```

Figure 11: Nibble shifter specification.

nanocycle time can be reduced to something near the propagation delay time of the multiplier chips plus the clock to output time of the multiplier register plus the set-up time of the intermediate result register. The disadvantages of this scheme are increased pipe latency, caused by the two extra levels of pipelining, and a high part count.

Yet another technique involves replacing one level of the pipe and one level of lookahead adders with carry-save adders between the partial product generators and the pipeline registers. Carry-save adders are used to implement a technique called 3-to-2 counting. As can be seen from the table below, any combination of three equally weighted bits can be recoded into a 2-bit field.

Inputs	Outputs
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1

Thus it is possible to reduce the three operands generated by the multiplication process (the high and low partial products and the 64-bit intermediate product) into only two operands which may then be summed together in a single lookahead ALU 3-to-2 recoding that requires no carry propagate logic and is therefore very fast. Due to the speed of 3-to-2 counters, only one level of pipelining is required, which results in both a reduced parts count and a reduced pipe latency. The technique is ideally suited for implementation with PALs.

In the APU floating point engine, 16 AmPAL16R6s, programmed as triple 3-to-2 counters, are used to reduce the three multiplication operands to two intermediate results. A logic diagram is shown.

The registered outputs of the PALs are connected to the input buses of the mantissa ALU, which is also used for floating point addition and subtraction. The mantissa ALU then calculates the next intermediate product in parallel with the partial products calculations occurring in the 8×8 multipliers. This intermediate product and the new partial products are then recoded by the 3-to-2 counter PALs to form the next pair of intermediate results. This process continues until the complete 56×56 product is generated. Hence, without adding pipe latency, the APU is able to accumulate partial products at a rate of 8×56bits every 112ns, which happily coincides with the basic nanocycle time of the machine.

The APU barrel shifter can perform left shift, right shift and rotate operations of 0 to 63 bits in a single microcycle. The barrel shifter is used mainly for floating point prescale and normalise operations. A block diagram of the barrel shifter and its associated control logic is shown in figure 6. The word rotater, nibble shifter, and bit shift and mask logic comprise the three stages necessary to implement the barrel shifter. The prescale, leading zero detect, and mask control logic comprise the logic required to control it.

The prescale logic converts the signed difference produced by the exponent arithmetic units as a result of the comparison of the two operand exponents into an absolute shift distance, which is then used to right shift (prescale) the mantissa of the smaller operand of a floating-point add or subtract operation. The leading zero detect logic determines the left shift distance required to produce a left-justified (normalised) result. The mask control logic is used to convert rotated data to shifted data by masking off the appropriate leading or trailing bits to implement right or left shifts.

Simple implementation

To implement the three level 64-bit barrel shifter of figure 6 in MSI requires the use of Am25S10 4-bit shifters (see figure 7). The first level is the word rotator which performs a circular rotate of 0, 16, 32 or 48 bits. Although implementation is simple, the MSI solution requires 16 packages. The second level is the nibble shifter, which is essentially identical to the word rotator but is wired to rotate 0, 4, 8 or 12 bits. The final stage of the barrel shifter requires not only bit rotate but also leading and trailing bit masking and sticky bit computation (ie, the logical OR of the masked out bits). An MSI solution would require not only the 16 packages of AM25S10s, used in each of the preceding levels, but also 16 packages of AND gates for masking, plus still another 16 packages of ANDs for the sticky bit computation. The control logic for performing the mask operation would probably require as much logic as the entire shift path. The more practical solution has usually been to build separate left and right shifters, 48 packages apiece, and not to implement a sticky bit at all.

The PAL implementation of a 64-bit rotator and shifter, with sticky bit computation, requires considerably fewer packages than a unidirectional MSI shifter.

The word rotator consists of eight identical PALs programmed as two four-bit rotators per package. The logic diagram of the word rotator is shown in figure 8 and the functional specification (using AMD's PLPL programming language) is shown in figure 9. The nibble shifter requires four Am25S10s and eight PALs programmed as 6-bit-wide 4-place shifters. The logic diagram and functional specification of a nibble shifter are shown in figures 10 and 11. The bit shift and mask logic requires sixteen PALs in the data path and two PALs in the control path.

Nearest neighbour masking

The technique used to implement the masking function required for shifting is called 'nearest neighbour masking' (US patent pending, Computervision Corp.). Each of the shift and mask PALs has an enable input from one of the mask control PALs. In addition, each shift and mask PAL is also connected to the enable inputs from its left and right-hand neighbours. The mask control PAL input determines if all four bits from the slice should be masked off. The enables from the adjacent slice determine if a PAL is at a shift boundary. Should only one of the neighbouring slices be disabled, then the shift and mask PAL will mask off from 0 to 3 of the bits adjacent to the disabled slice, depending on the bit rotation distance. In this way, the 64-bit masking operation can be implemented with only 16 control lines as opposed to at least 64 for an SSI/MSI solution.

The entire PAL barrel shifter requires 38 devices to implement 64-bit rotation, left shifting, right shifting and sticky bit accumulation. An MSI-based left/right shifter, without sticky bit computation, would require a minimum of 96 parts. □

With a new hardware description language, designers can define functions at the programmable logic level. Major benefits include faster turnaround and improved first-time results.

Logic-programming language enriches design processes

Programmable logic devices have long given designers unbeatable productivity advantages. No other semicustom logic technology can boast a faster turnaround: New designs can be created in minutes by programming, or "blowing," appropriate fuses. Even more important, there is no waste in this haste. Performance, flexibility, and cost are not penalized. Now imagine how much more beneficial programmable logic would be if it could draw on the assets of computer-aided design.

With the emergence of a set of high-level CAD tools for programmable logic devices, designers can translate logic into functional custom devices more simply and efficiently than ever. The nerve center of the package is a block-structured hardware description language called PLPL, for "programmable-logic programming language."

The beauty of PLPL lies in its multiple input formats, which permit different design approaches for a variety of design problems. The higher the level of the approach, the closer PLPL will come to directly specifying the desired function. Intermediate steps in the design process can be eliminated, along with the errors that might have been generated during those steps. The results are a quicker design cycle and a device that works the first time.

PLPL's input formats are quite versatile, ranging from a simple Boolean sum of products, to deMorgan's theorem, to high-level formats that support state machine and bus-oriented data-path functions. Earlier programmable-logic support software, such as PALASM software, has been limited to a sum-of-products Boolean entry.

Brad Kitson, Section Manager
Kevin Ow-Wing, Engineer
Product Planning and Applications
Advanced Micro Devices Inc.
901 Thompson Place, Sunnyvale, Calif. 94088

An overview of its breadth

The language can accept input specifications for all architectures based on the AND-OR configuration, including PROM, PAL, and PLA devices. At present, PLPL supports the AmPAL-22V10 and all AMD 20-pin PAL devices. Future AND-OR devices can be accommodated easily merely by updating the device data base. Thus, PLPL will not face obsolescence as new, more powerful programmable logic devices appear.

The PLPL package aids in other facets of design. With it, the designer can simulate, or verify, that an input specification will initially translate into a working device. He or she can also apply test vectors following programming to ensure that the device functions properly before placing it on a board or in a system.

The right stuff

Since PLPL was developed in the C language, it is compatible with a wide assortment of computers. Consequently, source-code versions will appeal to those desiring to run the CAD tools on their own computers or workstations, and a limited number of object-code versions will run on machines like the IBM personal computer and under operating systems like Unix.

Because of its block structure, PLPL can stand alone as a complete design package, or it can be integrated into a larger CAD system.

Yet PLPL and most CAD systems exhibit three main differences. First, PLPL is a device-oriented tool. Larger CAD systems, usually embodied in boxes called CAD workstations, are geared toward a system approach. Second, PLPL solves a very specific problem, something that a workstation's broader outlook more or less prohibits. Last, PLPL can provide crucial help during the definition phase of design. Most workstations begin at the board level and relegate device "design" to entries from a library.

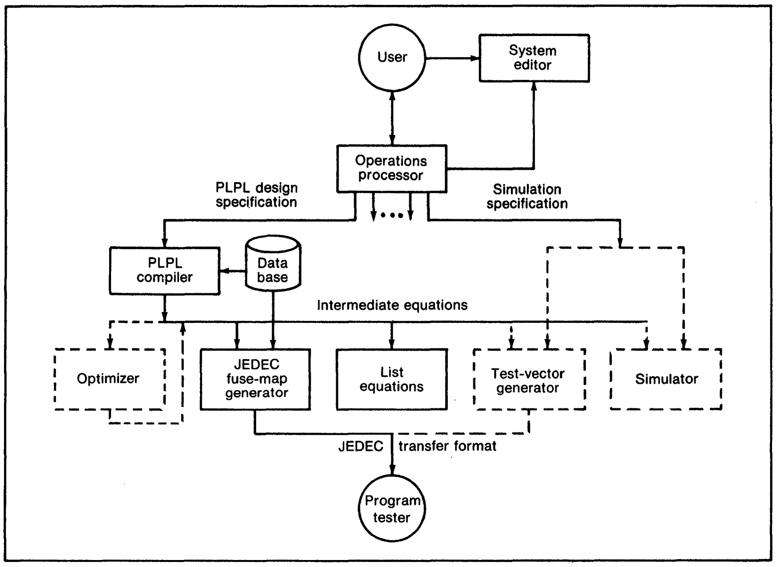
On a workstation, an engineer designs graphically by placing device representations from the libraries on a CRT screen and drawing connecting wires. One simulator and one test-vector generator usually are present, each aimed at solving board-level problems.

Integrating PLPL into a workstation creates a true synergism—the correlation between system levels and device levels is now possible, significantly reducing the opportunity for errors. The only requirement for integration is that the workstation accommodate hierarchical design—that is, the designer must be able to create his or her own device library with PLPL for subsequent use at the system level.

The PLPL structure

In a CAD environment, each PLPL design process—defining the problem and devising and verifying the solution—is governed by a separate program module. The modules are tied together in a top-down fashion.

Recall that PLPL itself is a block-structured language. It includes the high-level constructs IF ... THEN ... ELSE, CASE, and FOR. Macros, functions, constants, and variables also are part of PLPL, affording both clarity and documentation. At present, PLPL consists of four main modules: an operations processor, a compiler, a fuse-map generator, and a data base (Fig. 1). One other module, a simulator, is in



1. Five major software modules make up the structure of the programmable logic programming language (PLPL). The operations processor serves as the line between the designer and PLPL's program modules. Additional modules may be added to this basic package as they are developed.

development. The modular configuration of PLPL will simplify the addition of new functions to the software package in the future. Possible additions include product-term optimization and test-vector generation.

The operations processor is the glue that binds PLPL to the CAD environment. It is an interactive, menu-driven shell through which the designer can reach the other modules, as well as some device-specific utilities. With the operations processor, the user can consult a comprehensive help facility or perform a "shell escape." In the latter case, he or she temporarily exits the PLPL environment to execute such system operations as editing. The shell maintains its internal status in the meantime.

The PLPL compiler is the key member, as it converts design specifications written in PLPL into the form needed by the fuse-map generator and other modules, all the while flagging syntax and device limitation errors. (An error, for example, might indicate that a programmable logic circuit has an insufficient number of inputs to accommodate a particular logic equation.) The compiler is not tied to any particular device and can obtain all device-sensitive data from the data base.

Device simulation is based on test or function tables, created by the designer, that list input and expected output conditions. The PLPL simulator takes these inputs and combines them with the compiler's output to model the device's output behavior. It then compares the resulting output with the expected outputs and issues an error message if the results do not match. A test table may be concatenated, or linked, with the design input specification, or it may be kept in a separate file. The latter alternative enhances flexibility and allows more than one test table to be attached to a design.

The fuse-map generator accepts design details from the compiler and should be invoked following a successful simulation. It arranges the fuse pattern required for programming the device into a JEDEC transfer format—supported by all major suppliers of logic programmers. The module generates a fuse map that can be downloaded directly to a programmer.

The data base serves a threefold purpose. First, it stores all the architectural information associated with supported devices and supplies this data to the compiler and fuse-map generator. This arrangement makes the latter modules transparent to variations in device architectures. Next, the data base gladly accepts new devices, making the PLPL environment more powerful—all it takes is the addition of a description file to the data base. Third, the data base serves as a standard-cell library, thereby automating programmable logic design. Currently, it carries all AMD PAL devices.

The essence of the language

How does PLPL work in action? A good example of its capabilities is given through the implementation of an 8-bit shift register (Fig. 2). Input specifications, which are created

```
DEVICE SHFT8BIT (AMPAL22V10)
       "This is a simple example of an eight-bit
       shift register using the AmPAL22V10."
PIN    CLOCK = 1            RESET = 13            SELECT [1:0] = 2, 11
       RILO = 23            "Right shift Input, Left shift Output"
       LIRO = 14            "Left shift Input, Right shift Output"
       DATA [7:0]    = 3, 4, 5, 6, 7, 8, 9, 10
       Q_OUT [7:0]   = 22, 21, 20, 19, 18, 17, 16, 15;
DEFINE LOAD  = /SELECT [1] * /SELECT [0];     "load data"
       SHFTR =  SELECT [1] * /SELECT [0];     "shift right"
       SHFTL = /SELECT [1] *  SELECT [0];     "shift left"
       HOLD  =  SELECT [1] *  SELECT [0];     "hold data"
BEGIN
       IF (RESET) THEN ARESET ();
       IF (SHFTL) THEN ENABLE (RILO);
       RILO = Q_OUT [7];
       IF (SHFTR) THEN ENABLE (LIRO);
       LIRO = Q_OUT [0];
       Q_OUT [7] :=  LOAD  * DATA [7]   + SHFTR * RILO        +
                     SHFTL * Q_OUT [6] + HOLD  *  Q_OUT [7];
       Q_OUT [6] :=  LOAD  * DATA [6]   + SHFTR *  Q_OUT [7]  +
                     SHFTL * Q_OUT [5] + HOLD  *  Q_OUT [6];
       Q_OUT [5] :=  LOAD  * DATA [5]   + SHFTR *  Q_OUT [6]  +
                     SHFTL * Q_OUT [4] + HOLD  *  Q_OUT [5];
       Q_OUT [4] :=  LOAD  * DATA [4]   + SHFTR *  Q_OUT [5]  +
                     SHFTL * Q_OUT [3] + HOLD  *  Q_OUT [4];
       Q_OUT [3] :=  LOAD  * DATA [3]   + SHFTR *  Q_OUT [4]  +
                     SHFTL * Q_OUT [2] + HOLD  *  Q_OUT [3];
       Q_OUT [2] :=  LOAD  * DATA [2]   + SHFTR *  Q_OUT [3]  +
                     SHFTL * Q_OUT [1] + HOLD  *  Q_OUT [2];
       Q_OUT [1] :=  LOAD  * DATA [1]   + SHFTR *  Q_OUT [2]  +
                     SHFTL * Q_OUT [0] + HOLD  *  Q_OUT [1];
       Q_OUT [0] :=  LOAD  * DATA [0]   + SHFTR *  Q_OUT [1]  +
                     SHFTL * LIRO      + HOLD  *  Q_OUT [0];
END.
```

2. An input specification for an 8-bit shift register shows how PLPL works. A pin list assigns symbolic names to help define functions.

with the system editor, require at least a heading, a pin list, and an equation section. Those familiar with PALASM software will recognize that the equation section is almost identical but that the heading and the pin list are quite different. The optional definition section and comment strings are not possible with PALASM software.

The heading, beginning with the key word, DEVICE, gives a title to the design, SHFT8BIT, and identifies the device in question, AMPAL22VIO. This format is similar to that of a function or subroutine called in high-level software, with the title analogous to the name of the function, and the device to the argument being passed to the function.

Shown between the heading and the pin list is a comment string, which begins and ends with double quotes. The compiler ignores all text within the quotes. In fact, comment strings can appear anywhere in a specification.

After the comment comes the pin list, indicated by key word PIN, which assigns symbolic names within the device to help describe a pin's function. Generally, the name of the circuit appears before its assigned pin, as in CLOCK = 1. Notice that the equal sign links the user-specified symbolic name with the pin number. Pin names and numbers may appear in any order, with white space between the key word and the first entry and between succeeding entries. A semicolon marks the end of the list.

```
DEFINE LOAD  = /SELECT [1]  *  /SELECT [0];
       SHFTR =  SELECT [1]  *  /SELECT [0];
       SHFTL = /SELECT [1]  *   SELECT [0];
       LOAD  =  SELECT [1]  *   SELECT [0];

       Q_OUT [9] = RILO;    Q_OUT [0] = LIRO;

VAR    PIN_NUM;

BEGIN
       IF (RESET) THEN ARESET ();
       IF (SHFTL) THEN ENABLE (RILO);
       RILO = Q_OUT [8];

       IF (SHFTR) THEN ENABLE (LIRO);
       LIRO = Q_OUT [1];

       FOR (PIN_NUM = 1 TO 8)
           Q_OUT [PIN_NUM] := LOAD  * DATA [PIN_NUM]              +
                              SHFTR * Q_OUT [PIN_NUM PLUS 1]  +
                              SHFTL * Q_OUT [PIN_NUM MINUS 1] +
                              HOLD  * Q_OUT [PIN_NUM];
END.
```

3. In PLPL, loop constructs like FOR **present designers with an easy way of evaluating pin vector expressions for, say, the 8-bit shift register.**

If desired, multiple pins may be grouped in a single statement to designate a pin vector. For example, in the statement:

Q_OUT[7:0] = 22, 21, 20, 19, 18, 17, 16, 15

the symbolic pin vector name is concatenated with the numbers for each symbolic pin enclosed in brackets. Within the brackets, the leftmost number represents the most significant bit and the right-most number the least significant bit.

The colon defines a sequential set of numbers, and the comma concatenates the numbers. Colons and commas can be combined for both symbolic and actual pin numbers. For example, another way to express the above pin vector is:

Q_OUT[7:4,3,2,:0] = 22, 21, 20:15

or most succinctly:

Q_OUT[7:0] = 22:15

This technique adds flexibility, clarity, and conciseness to input specifications. Instead of having to deal with each device pin individually, all pins can be considered a single unit (a set of pins that interfaces with a bus is a good example). This feature also provides a clear and powerful way of defining state machines, with pin vectors representing the various states.

The next section in Fig. 2 defines four macros, beginning with the key word DEFINE. Each macro definition terminates with a semicolon, and white space is allowed for formatting purposes. By macro is meant a single symbolic name that represents a more complicated expression: The significance of LOAD is more immediately apparent to the designer than is /SELECT [1] */SELECT [0]. Macros can also be viewed as a form of documentation—in fact, the comments after each macro definition probably could be removed without loss of understanding.

Macros make specifications more concise in the equation section. Perhaps less obvious, they reduce the number of typing errors because fewer key strokes are required. Furthermore, design changes do not have to be made repeatedly throughout the equation section.

That section, which defines the function of each output, starts with the key word BEGIN and terminates with the key word END. (note the period). The general format for Boolean equations is:

```
PIN NAME {:=} EXPRESSION:
```

An example of the Boolean format is:

```
Q_OUT[7] := LOAD * DATA [7] +
            SHFTR * RILO +
            SHFTL * Q_OUT [6] +
            HOLD * Q_OUT [7] ;
```

An expression is a sequence of pin names (or their complements), separated by operators. As before, pin names are symbolic names and are defined by the pin list. The operator := defines a sequential expression, that is, an expression for a registered output. The = operator defines an expression for a combinatorial output. Other operators include asterisk, plus, slash, and semicolon symbols, which respectively stand for AND (product), OR (sum), NOT (complement), and the expression terminator.

The IF conditional statement, similar to that of other high-level programming languages, activates enable and asynchronous reset functions that are specific to the device being programmed. The first function is called with the name, ENABLE, followed by parentheses containing names of the outputs that should be enabled. Empty parentheses signal a default of all outputs (Fig. 3). Similarly, the asynchronous reset is called with the name, ARESET.

```
DEFINE  LOAD   = 0;
        SHFTR  = 1;
        SHFTL  = 2;
        HOLD   = 3;
BEGIN
        IF (RESET) THEN ARESET ();

        IF (SHFTL) THEN ENABLE (RILO);
        RILO = Q_OUT [7];

        IF (SHFTR) THEN ENABLE (LIRO);
        LIRO = Q_OUT [0];

        CASE (SELECT [1:0] )
        BEGIN
              LOAD)   Q_OUT [7:0]  :=  DATA [7:0] ;
              SHFTR)  Q_OUT [7:0]  :=  RILO, Q_OUT [7:1];
              SHFTL)  Q_OUT [7:0]  :=  Q_OUT [6:0], LIRO:
              HOLD)   Q_OUT [7:0]  :=  Q_OUT [7:0] ;
        END;
END.
```

4. Another high-level PLPL construct, CASE, **makes decisions by evaluating expressions based on comparisons between variables and constants.**

The format of the IF condition with a function call is IF (CONDITION) THEN FUNCTION_NAME (PIN NAMEs);. The expression PIN NAMEs in the above listing is as previously defined, and FUNCTION_NAME is, of course, just that. CONDITION can be an expression, as defined earlier, or a relational test, such as SHFTL /= 0 (/= stands for "not equal to") or SELECT[1:0]<3.

A function call need not be the result of a true IF condition. Any expression or block of expressions can be executed. (A function call is a type of expression.) Also, if the condition is false, the key word ELSE can control the execution of another set of expressions. The general format of the IF ... THEN ... ELSE condition, which is similar to that of other high-level languages, is:

```
IF (EXPRESSION) THEN BEGIN
        EXPRESSION(S);
        END;
ELSE BEGIN
        EXPRESSION(S);
        END;
```

If only one expression is executed as a result of a true condition, the key words BEGIN and END; can be omitted. Also, the entire construct, including the expression blocks it controls, is considered to be a single expression (to determine the necessity of including BEGIN and END;).

Other high-level constructs

Besides IF ... THEN ... ELSE, PLPL offers many other high-level constructs, namely, the FOR construct for controlled loops and the CASE statement for making decisions. Both permit concise, understandable, and self-documenting specifications to be written.

The design of the 8-bit shift register clearly shows the benefits derived from the FOR construct (Fig. 3 again). The shift register now takes only one statement to define, instead of eight. By providing user-controlled loops, FOR makes it easy to evaluate expressions, especially those incorporating pin vectors. The general format of a FOR loop is:

```
FOR (VARIABLE=/CONST_EXPRESSION
TO CONST_EXPRESSION)
    BEGIN
    EXPRESSION(S):
    END;
```

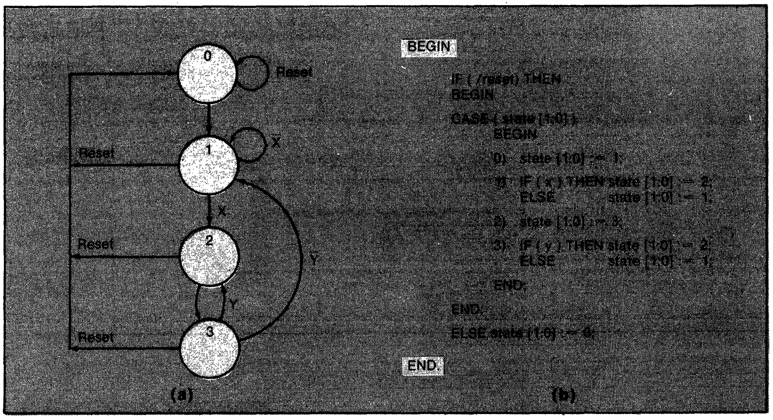
where VARIABLE is the control index of the loop and is defined in a section defined by the key word VAR. As with constants, the declaration of a variable must appear before it is used. Initially, the variable receives a value equal to that of CONST_EXPRESSION, which is evaluated once. Each time the loop is entered (including the initial entry), the current value of VARIABLE is tested against CONST_EXPRESSION, whose constant value, when reached, terminates the loop. The key word in this construct is FOR. As in many cases, BEGIN and END; are omitted.

The other high-level construct, CASE, is similar to the switch statement in C and the case statement in Pascal. CASE transfers control to one of several blocks of statements, depending on the value of a variable, such as a pin vector. A pin vector's value is its numerical equivalent (if SELECT[1] = 1 and SELECT [0] = 0, the value of the pin vector SELECT[1:0] = 10 in binary or 2 in decimal).

CASE tests the value of the variable against a list of constant values, called a case list, defined by the designer. If the variable matches a constant, the block of expressions associated with the constant is executed. A default block of expressions may be provided by calling on the key word DEFAULT in the list. If no default exists and none of the constants matches the variable, no expressions are evaluated.

Incorporating CASE into the shift register design illustrates its power and resulting clarity (Fig. 4). The variable is the pin vector SELECT[1:0], which is compared with four predefined constants—LOAD, SHFTR, SHFTL, and HOLD—spelled out in the DEFINE section. Notice that CASE employs the same control expressions that were explicit in the macro definition of the original specification. Thus the actual functions—not the implementation of these functions—have been specified.

Designing a state machine is another good demonstration of how CASE shines (Fig. 5). Each state is directly implemented with a different CASE, each of which contains the statements needed to perform the next state transition. The IF statements express the conditional next-state transitions. As a result the state diagram can be directly translated into a PLPL specification. The construct allows the design



```
BEGIN
        IF ( /reset) THEN
        BEGIN
        CASE ( state [1:0] )
            BEGIN
            0)  state [1:0] := 1;
            1)  IF ( x ) THEN state [1:0] := 2;
                ELSE          state [1:0] := 1;
            2)  state [1:0] := 3;
            3)  IF ( y ) THEN state [1:0] := 2;
                ELSE          state [1:0] := 1;
            END;
        END;
        ELSE state [1:0] := 0;
END.
```

5. Used in state machine design, the CASE **statement merges with the** IF **condition so that a state diagram (a) can be directly translated into a PLPL specification (b).**

to be partitioned into sections that are simple to deal with. In fact, a designer experienced with PLPL need not bother with the state diagram at all.

Pulling it all together

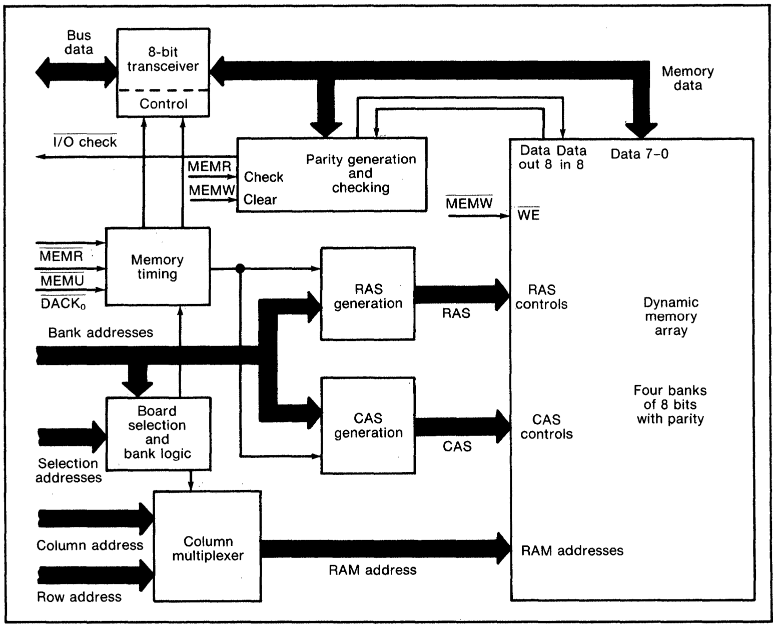
One formidable design challenge—a memory system—puts PLPL in its best light. Dynamic memory arrays usually leave little board space for logic, which nevertheless must be there and must produce difficult timing signals. One solution is to use PAL circuits for the logic functions, because they occupy board space efficiently and can generate the required signals.

Suppose the memory system in question is a 256-kbyte memory board for the IBM personal computer. The first step involves identifying the various logic and function blocks within the memory system (Fig. 6). Once identified, the various functions need to be partitioned into PAL devices. This step requires counting the inputs and outputs of each block to see how much of a given PAL device will implement that function (see the table at left). For example, the data bus transceiver block cannot be implemented with a PAL device—too many outputs.

Moreover, the amount of logic for a given function should be estimated to see how many AND-OR structures are required. The parity function, for example, takes two AND-ORs. For this reason, and because its nine inputs are not shared by any other PAL circuit, it is a likely

Input and output requirements of function blocks

Function	Inputs	Outputs
Memory timing	4	5
RAS generation	5	4
CAS generation	5	4
Parity generation and checking	11	2
Board and bank selection logic	8	1
Address multiplexer	17	8
Data bus transceiver	18	16



6. The first step in applying PLPL involves partitioning a design, here a memory board, into functional blocks and then assigning all or part of the functions to various programmable logic devices. By sharing inputs, the functions can be merged into three PAL circuits.

candidate for implementation instead by a parity generator. The remaining functions can each be implemented with a single AND-OR, except for the memory timing section, which carries a delay line.

Shared inputs are a key to merging separate functions into one PAL device. The bank address is shared among the logic blocks for RAS generation, CAS generation, and board and bank selection. The savings in pin count allows RAS, CAS, and bank selection to be combined in a single 24-pin device. Another PAL device carries the lower seven bits of the address multiplexer. The uppermost address multiplexer bit and all of the remaining logic, minus the parity generator, are partitioned into yet another PAL circuit. Now PLPL can step in. (An alternative method first defines the functional blocks with PLPL, then partitions the defined blocks into the PAL devices.)

```
DEVICE     RAS_CAS (PAL 22V10)
"
       PLPL file for:  RAS timing
                       CAS timing
                       Valid Access signal
"
PIN
       /Memory_Cycle_0   = 1          /Memory_Cycle_20 = 2
       /Memory_Cycle_100 = 3
       /DACK0 = 4                     /Board_Active = 5
       Address [16:17] = 6:7           Bank_Active [1:4] = 8:11
       /CAS [1:4] = 14:17             /RAS [1:4] = 18:21
       /Valid_Access = 22;
DEFINE
       Refresh_Cycle = DACK0;
BEGIN
       " Valid Access Generation "
       IF ( Board_Active * /Refresh_Cycle * Memory_Cycle_0 )
           THEN CASE ( Address [17:16] )
                  BEGIN
                         0)  Valid_Access = Bank_Active [1];
                         1)  Valid_Access = Bank_Active [2];
                         2)  Valid_Access = Bank_Active [3];
                         3)  Valid_Access = Bank_Active [4];
                  END;
       " RAS Generation "
       IF ( Refresh Cycle * Memory_Cycle_0 )
           THEN  RAS [4.1] = Bank_Active [4:1];
           ELSE IF ( Board_Active * Memory_Cycle_0 )
                  THEN CASE ( Address [17:16] )
                         BEGIN
                                0)  RAS [1] = Bank_Active [1];
                                1)  RAS [2] = Bank_Active [2];
                                2)  RAS [3] = Bank_Active [3];
                                3)  RAS [4] = Bank_Active [4];
                         END;
       " CAS Generation "
       IF ( Board_Active * /Refresh_Cycle )
           THEN IF ( Memory_Cycle_20 * Memory_Cycle_100 )
                  THEN CASE ( Address [17:16] )
                         BEGIN
                                0)  CAS [1] = Bank_Active [1];
                                1)  CAS [2] = Bank_Active [2];
                                2)  CAS [3] = Bank_Active [3];
                                3)  CAS [4] = Bank_Active [4];
END.
```

7. In the memory board of Fig. 6, RAS and CAS signals and bank selection are the responsibility of one PAL circuit, as shown in the PLPL specification for that device.

The PLPL specification for the first device defines bank selection, RAS, and CAS—all by CASE statements (Fig. 7). Bank selection occurs if the board has been selected (Board_ Active), the refresh cycle (/Refresh_Cycle) has been selected, and the delay line has been triggered (Memory_Cycle_0). It is a valid access if the bank selection switch, decoded by Address[17:16], is high. RAS and CAS generation occur in much the same way—the board is selected and timing controlled by a phase of the delay line. A DMA channel on the personal computer controls refreshing; when DACKO is low (Refresh_Cycle), the RAS outputs all are activated, depending on the settings of the bank selection switches.

The PLPL specification of the address multiplexer defines the function with a single IF statement. It can be read if the delay line phase, Memory_Cycle_60, is high. Then, the column-address bus is selected; otherwise, the row address bus must be selected.

The PLPL specification for the last PAL device, which contains the remaining glue logic for the design, includes the memory timing logic that triggers the delay line and controls the data bus transceiver. In addition, it holds the parity generation and checking logic, the board selection logic, and the uppermost bit of the address multiplexer. These functions are easily implemented, thanks to PLPL's high-level descriptions.

The last step in the design process is to compile the three devices. If syntax errors occur, the design must be corrected to obtain a successful recompilation. All three are then simulated; if the devices have been completed correctly, the fuse-map generator can be invoked, with its outputs fed to a PAL programmer.□

Brad Kitson, Dave Laws,
Warren Miller

Logik-Baustein kombiniert PAL- und Gate-Array-Technik

Programmierbare Logikbausteine der zweiten Generation stellen dem Entwickler eine Kombination der Eigenschaften von PAL- und Gate-Array-Bausteinen zur Verfügung. Kosteneffektivität, ein höherer Grad an Funktionalität sowie eine kürzere Entwicklungszeit im Vergleich zu anderen Logik-Bauelementen lassen sich mit diesem Baustein, der die Typenbezeichnung AmPAL22V10 trägt, erreichen. Eine Schlüsselrolle für die verbesserten Eigenschaften spielt die „programmierbare Architektur", mit deren Hilfe ein Entwickler den Aufbau des Bausteines festlegen kann.

PAL-Bausteine sind äußerst flexible Elemente für die Entwicklung, weil der Anwender seine Logikschaltung im eigenen Haus auf der Grundlage preiswerter, allgemein erhältlicher ICs selbst fertigen kann. Gate-Arrays bieten dagegen einen höheren Integrationsgrad als heutigen PAL-Bausteine, erfordern aber längere Entwicklungszeit und größeren Kostenaufwand. Trotzdem bieten Gate-Arrays mehr Möglichkeiten, eine spezialisierte Schaltung zu realisieren, weil sie auf elementaren Logikstrukturen basieren. Wenn Gate-Arrays zur Implementierung komplexer Funktionen herangezogen werden, erfordern sie eine vertikale Logikstruktur, die diese Bauelemente langsam machen können. PAL-Bausteine sind ähnlich wie Speicher matrixartig aufgebaut. Obwohl für die Programmierung gewisse zusätzliche Schaltungsteile vorhanden sind, ist die Dichte eines PAL-Bausteines mit dem eines Gate-Arrays durchaus vergleichbar.
Der Baustein AmPAL22V10 kann als Gate-Array aufgefaßt werden, das mit Hilfe von Durchschmelz-Verbindungen zu programmieren ist. Unter Beibehaltung aller Vorteile eines PAL-Chips läßt sich mit einem solchen IC eine Schaltung von 500...1000 Gattern ersetzen. Man schätzt, daß 80...90 % der heutigen Gate-Arrays in Schaltungen dieser Dichte Verwendung finden.

Außer den derzeitigen schaltungstechnischen Lösungen mit PAL-Chips ersetzt der Typ AmPAL22V10 funktionsmäßig so gut wie alle SSI/MSI-Logikbausteine.

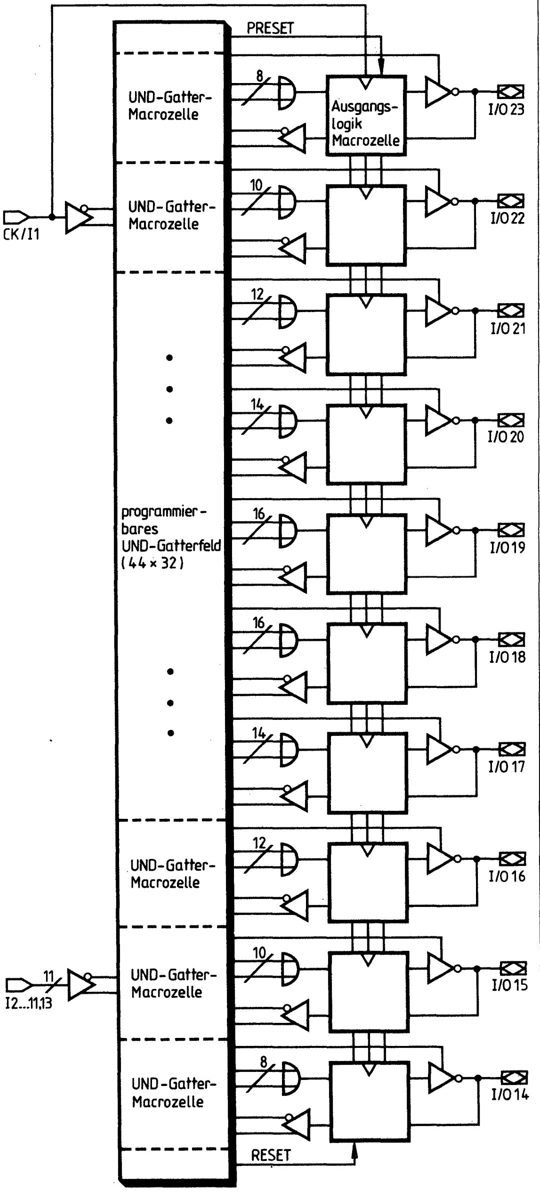


Bild 1. Blockschaltung des programmierbaren Logik-Bausteins AmPAL22V10. Bei diesem Chip handelt es sich um eine Kombination aus Gate-Array- und PAL-Technik. Besonderes Merkmal sind die Ausgangs-Logik-Makrozellen, die einzeln in vier verschiedenen Konfigurationen zu programmieren sind

Aufbau des Chips

Bild 1 zeigt die Blockschaltung des AmPAL22V10. Alle 22 Eingänge liegen an einem programmierbaren UND-Gatterfeld mit über 5800 durchschmelzbaren Verbindungen. Dieses Gatterfeld besteht aus 120 UND-Verknüpfungen mit jeweils 44 Eingängen, deren Ausgänge an zehn ODER-Gatter angeschlossen sind. Auf diese Weise lassen sich die üblichen Logikfunktionen implementieren, die in Form Boolescher Produktsummen gegeben sind. Im UND-Array sind auch zehn UND-Gatter zur Ausgangssteuerung mit jeweils 44 Eingängen, ein gemeinsames asynchrones Reset-UND-Gatter mit 44 Eingängen sowie ein synchrones Reset-UND-Gatter mit 44 Eingängen vorgesehen. An den ODER-Gattern befinden sich die zehn Ausgangs-Logik-Makrozellen. Diese können individuell in ihrer Funktion programmiert werden.

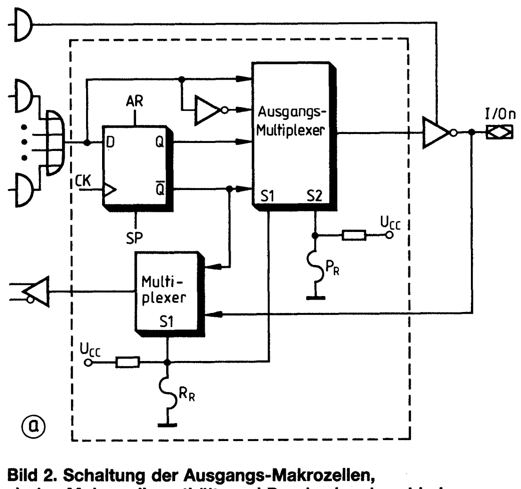
Der Baustein befindet sich in einem 24poligen DIL-Gehäuse mit 0,3 Zoll Breite, wobei die Stiftbelegung so gewählt wurde, daß die Eingänge sich auf der linken und die Ausgänge auf der rechten Seite befinden. Zehn dieser Anschlüsse sind bidirektional (14...23); auch sie lassen sich individuell programmieren als Ausgang, Eingang oder dynamisch steuerbare E/A-Anschlüsse. Stift 1 dient zum Anschluß des Taktsignals oder als spezieller Eingang.

Der Baustein stellt die Kombination aus 20 programmierbaren Makrozellen mit einer festen Logik dar. 22 Eingangspuffer geben ihren direkten und invertierten Ausgang auf ein UND-Gatterfeld, das in zehn UND-Gatter-Makrozellen eingeteilt ist. In bezug auf die Komplexität lassen sich die zehn Makrozellen mit einer Schaltung von 8...16 logischen UND-Gattern vergleichen, die um eine Ausgangs-Enable-Schaltung ergänzt ist. Die Anzahl der UND-Gatter nimmt mit steigender Nummer der Makrozelle zu. Es ergibt sich eine Verteilung der Gatterzahl von 8, 10, 12, 14, 16, 16, 14, 12, 10, 8. Im Mittel befinden sich zwölf Gatter in einer Makrozelle.

Die variable Verteilung der UND-Gatter kommt insbesondere der Implementierung von Zählfunktionen, Exklusiv-ODER-Verknüpfungen sowie Maschinen für komplexe Zustände entgegen. So verfügt z. B. ein 10-Bit-Binärzähler über lediglich ein UND-Gatter auf dem LSB, für jedes weitere Bit kommt ein weiteres Gatter hinzu, so daß beim MSB zehn UND-Gatter erforderlich sind. An den Ausgängen der UND-Gatter jeder Makrozelle liegt eine ODER-Verknüpfung mit 8...16 Eingängen, deren Signale zu einer von zehn prorammierbaren Ausgangs-Makrozellen weitergeleitet wird. Über die Ausgangs-Enable-Gatter und Tri-State-Ausgangspuffer sind die E/A-Anschlüsse verbunden.

Programmierbare Architektur

Wichtigstes Merkmal des AmPAL22V10 ist die Programmierbarkeit der Ausgangs-Makrozellen (*Bild 2a*). Diese besteht aus drei Logikfunktionen: einem D-Flipflop, das mit ansteigender Flanke getriggert wird und einem 4-zu-1-Multiplexer für die Auswahl des Ausgangspfades sowie einem 2-zu-1-Multiplexer für die Auswahl des Rückkopplungs-Pfades. Zur Auswahl der Ausgangskonfiguration befinden sich in dieser Stufe zwei Durchschmelzverbindungen („Rn“ und „Pn“), mit deren Hilfe sich vier verschiedene Konfigurationen pro-



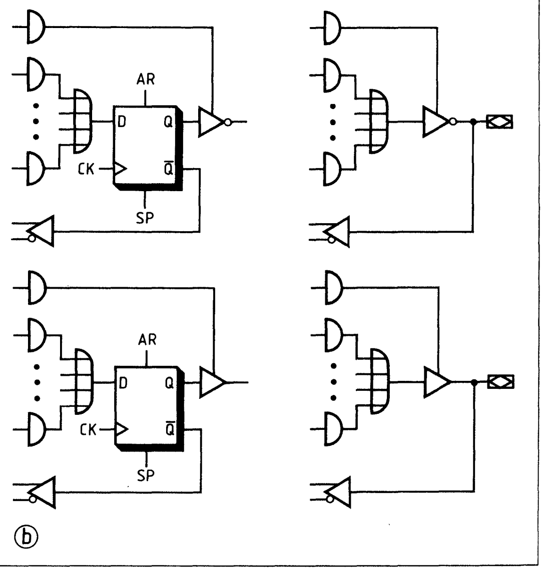


Bild 2. Schaltung der Ausgangs-Makrozellen,
a) eine Makrozelle enthält zwei Durchschmelzverbindungen (RR, PR), die der Programmierung dienen,
b) die vier möglichen Konfigurationen der Ausgangszelle

grammieren lassen: aktiv Low mit Register, aktiv High mit Register, aktiv Low kombinatorisch und aktiv High kombinatorisch (Bild 2b).

Zuverlässige Technologie

Der Baustein AmPAL22V10 wird mit Hilfe des bipolaren oxidisolierten Prozesses mit der Bezeichnung „IMOX" hergestellt. Es entstehen dabei Strukturen, die denen der Advanced-Schottky-Logik gleichen. Die kleinsten Abmessungen sind die Emitter von 3,3 × 3,3 μm^2, die Metallbahnen haben eine Breite von 7,5 µm bzw. 11 µm. Bei diesem Aufbau erreicht man bei den schnellsten Ausführungen des AmPAL22V10 Verzögerungen zwischen Ein- und Ausgang plus Einstellzeit von 25 ns, in bezug auf den Takt ist ein Ausgangssignal um 15 ns verzögert. Schon während des Herstellungsprozesses erfolgt eine vollständige Prüfung aller Gleich- und Wechselspannungs-Parameter.

Für die Durchschmelz-Verbindungen kommt die Platin-Silikat-Technologie zur Anwendung, die Programmierzeiten beschleunigt und die Zuverlässigkeit des Bauelementes erhöht. Diese Bausteine können mit Hilfe eines einfach zu implementierenden Algorithmus auf einem breiten Spektrum kommerziell erhältlicher Geräte programmieren. Platin-Silikat als Material wählte man aus, weil sich dabei ein genau steuerbarer Schmelzvorgang erreichen läßt, der große nichtleitende Unterbrechungen erzeugt, die auch auf lange Sicht zuverlässig sind. Inzwischen liegen Erfahrungen mit 43 Mrd. Teststunden für die Durchschmelzverbindungen vor, ohne daß bei dieser Struktur ein einziger Ausfall aufgetreten ist.

Anwendungsbeispiel

Eines der wichtigsten Probleme beim Entwurf von Mikroprozessorsystemen ist der Anschluß von Peripherie-Ports an den Prozessor. Üblicherweise müssen diese Ports intelligent genug sein, um Zwei-Ebenen-Handshaking-Protokolle bearbeiten zu können, ohne daß der Prozessor damit belastet wird. Interrupts oder DMA-Aufrufe dienen dazu, die Aktionen vom Mikroprozessor auszulösen, so daß Polling-Operationen, die nur auf Kosten der Leistung gehen, überflüssig sind.

Als Beispiel für die Anwendung des AmPAL22V10 ist in *Bild 3* ein SCSI-Port (Small Computer System Interface) dargestellt. Dieses besteht aus einem bidirektionalen 8-Bit-Datenport (DB7...0), drei Steuerausgängen (ACK, SEL und RST) sowie fünf Steuereingängen (REQ, C/D, MSG, BSY und I/O). RST ist ein Signal, das auf Low-Pegel aktiv ist und zum Zurücksetzen der SCSI-Ports benutzt wird. Es geht automatisch auf Low-Pegel, wenn /WR und /RD gleichzeitig auf Low-Pegel sind (Systemreset). Mit Hilfe des SEL-Ausganges läßt sich eine gewünschte SCSI-Einheit des Ports adressieren. Das Select-Kommando aktiviert das SEL-Signal und gibt den im DB-Latch gespeicherten Wert auf den SCSI-Datenfluß aus. Dieser Wert wiederum wählt einen der acht möglichen SCSI-Bausteine aus. Abgeschlossen wird das Select-Kommando mit Hilfe des BSY-Signals vom Port. ACK und REQ sind die beiden Handshake-Signale, mit denen der Rest der Daten- und Steuertransfers erfolgt. Der angewählte SCSI-Baustein erkennt, ob der Controller ein Kommando erhalten möchte oder einen Datentransfer vornehmen will. Darauf unterbricht der Controller mit Hilfe eines Interrupts den Programmablauf des Prozessors, worauf der Prozessor das Statusregister liest, um zu erkennen, welche Art Anfrage vorliegt. Nach dem Lesevorgang wird der Interrupt wieder aufgerufen. *Tabelle 1* zeigt die Definitionen der Schreib/Lese-Register, *Tabelle 2* die Codierung der Bus-Instruktionen.

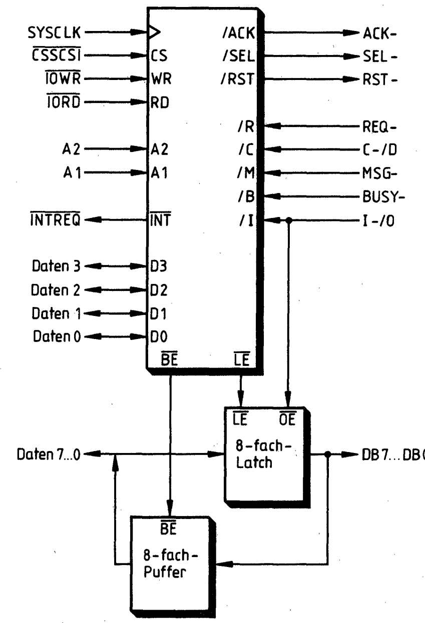


Bild 3. Blockschaltung eines SCSI-Controllers (Small Computer System Interface). Ein Baustein vom Typ AmPAL22V10 reicht aus, um alle Funktionen dieses Interface-Controllers auszuführen

Tabelle 1. Definition der Lese-/Schreib-Register

A2	A1	/RD	/WR
0	0	DB7...DB0	DB7...DB0
0	1	X	DB7...DB0, Select
1	0	C-/D, MSG-, BUSY-, I-/O, Löschen von INTRQ bei Lesen (Ausgabe ACK)	X
1	1	ACK-, SEL-, RST- REQ-	ACK-, SEL-, RST-, REQ-

```
/BE  =     /CS*/RD*/A2*/A1      ;READ DATA PORT

/LE  =     /CS*/WR*/A2          ;WRITE DATA PORT

IF(/CS*/RD*A2)  D3    =  /A1*/C      +
                         A1*/ACK

IF(/CS*/RD*A2)  D2    =  /A1*/M      +
                         A1*/SEL

IF(/CS*/RD*A2)  D1    =  /A1*/B      +
                         A1*/RST

IF(/CS*/RD*A2)  D0    =  /A1*/I      +
                         A1*/R

INTREQ     :=  /R*(/I*/C*/M     +                    ;COMMAND COMPLETE
                   /I*/C* M     +                    ;ERROR STATUS AVAILABLE
                   /I* C* M     +                    ;DATA AVAILABLE
                   I*/C* M      +                    ;COMMAND REQUEST
                   I* C* M)                          ;DATA REQUEST

ACK        :=  /CS*/WR*A2*A1*D3               +  ;WRITE FROM CPU
               /R */ACK*/(/CS*/WR*A2*A1)      +  ;HOLD, CLEAR ON REQUEST
               /R */CS*/RD*A2/A1                 ;ASSERT ACK ON COMMAND,
                                                 ;CLEAR ON REQUEST

SEL        :=  /CS*/WR*A2*A1*D2               +  ;WRITE FROM CPU
               B *SEL*/(/CS*/WR*A2*A1)        +  ;HOLD, CLEAR ON BUSY

RST        :=  /RD*/WR                           ;SYSTEM RESET
               /CS*/WR*A2*A1*D1               +  ;WRITE FROM CPU
                RST*/(/CS*/WR*A2*A1)             ;HOLD
```

Bild 4. Diese logischen Gleichungen fassen die Funktionen des SCSI-Controllers zusammen. Die logischen Verknüpfungen im AmPAL22V10 sind so zu programmieren

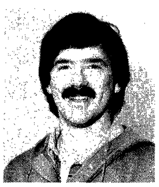
Obwohl es sich hierbei um die Lösung eines speziellen Problems handelt, zeigt dieses Beispiel, daß man eine solche Schaltung mit programmierbaren Logikbausteinen implementieren kann. Dann muß man zunächst überprüfen, ob ein AmPAL22V10 über die erforderlichen Ressourcen zur Lösung der Anwendungsaufgabe verfügt. Dies sind in erster Linie die Zahl der Ausgänge und Eingänge. Im Fall des SCSI-Bausteines sind das zehn Ausgänge, von denen die vier Daten-Ausgänge kombinatorisch sowie bidirektional und der Rest mit Register ausgerüstet sein müssen. Die zwölf übrigen Anschlüsse des Bausteins sind Eingänge. Davon erfordert ein SCSI-Controller elf, so daß in der Hinsicht keine Probleme auftauchen.

Jetzt müssen die Logik-Gleichungen aufgestellt werden, die die Funktion des SCSI-Controllers festlegen (*Bild* 4). /BE und /LE sind die üblichen Puffer- und Latch-Freigabe-Signale für Lesen und Schreiben von Daten. Die Gleichungen ergeben sich direkt aus der Registerdefinition in Tabelle 1. Als nächstes werden die Ausgangsfunktionen für die Daten bestimmt. Wenn der interne Status und die Steuerbits zu lesen sind, muß der Datenbus freigegeben sein. Alle Auswahlleitungen für die Status-/Steuerbits sind auf die Datenausgänge geschaltet. Das Interrupt-Request-Register ist gesetzt, wenn über den SCSI-Bus eine gültige Anfrage vorliegt. Bis der Controller dem SCSI-Baustein das ACK-Handshake-Signal gibt, bleibt der Interruptzustand aktiv. Dann schaltet REQ auf High-Potential, wodurch der Interrupt beendet wird. ACK wird automatisch gesendet, wenn REQ aktiv ist, und der Controller liest das Statuswort. SEL kann von der CPU geschrieben werden und ist automatisch aktiviert, wenn ein Select-Kommando ausgegeben wird. SEL wird deaktiviert, wenn BSY auf Low-Potential umschaltet. Wenn /RD und /WR gleichzeitig auf Low-Potential liegen (während des System-Reset) ist RST aktiviert. Dieses Signal kann auch von der CPU beschrieben werden, damit Reset-Operationen für das SCSI allein möglich sind.

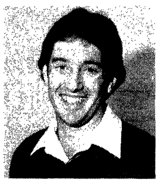
Tabelle 2. Codierung der SCSI-Bus-Instruktionen

I-/O	C-/D	MSG-	Definition
0	0	0	Kommando vollständig
0	0	1	Fehlerstatus verfügbar
0	1	0	–
0	1	1	Daten verfügbar
1	0	0	–
1	0	1	Kommando-Aufruf
1	1	0	–
1	1	1	Daten-Aufruf

Beim SCSI-Controller handelt es sich um einen relativ einfachen Peripheriebaustein. Die überzähligen Anschlußstifte des 24poligen Gehäuses können auch für andere Zwecke Anwendung finden. Auch wesentlich kompliziertere Mikroprozessor-Peripherieeinheiten können in einem einzigen Baustein vom Typ AmPAL22V10 untergebracht werden, z. B. eine Fließkomma-Normierungs-Einheit für Wortbreite oder ähnliche Funktionen.



Warren Kenneth Miller, Jr. studierte an der University of California in Berkeley (Master of Engineering in Elektrotechnik und Computerwissenschaften). Davor war er bereits bei AMD in Produktplanung für kundenspezifische Logik tätig. Jetzt gehört er zur AMD-Ingenieurabteilung in Sunnyvale, Ca.
Hobbys: das Dart-Spiel, Science-Fiction-Romane und der Besuch von Flughäfen.



Bradford Scott Kitson ist Bereichsleiter in der Abteilung für Produktplanung und Applikation für programmierbare Logikbausteine bei AMD in Sunnyvale, Ca. Seinen BS/EECS erhielt er an der Universität von Californien in Berkeley.

David A. Laws studierte an der Universität Hull (England) Physik, danach erhielt er in Birmingham das Certificate in Industrial Administration und besuchte noch das College of Advanced Technology in England. Heute ist er Managing Director für programmierbare Logikschaltungen bei AMD. Bevor David A. Laws zu dieser Firma kam, war er bei Fairchild, Litronix und Signetics.

Second-Generation Programmable Logic Devices Extend Design Capabilities

by

Jenny Yee
Applications Engineer
Advanced Micro Devices, Inc
9Ø1 Thompson Place
Sunnyvale, California 94Ø88

Introduction

The flexibility and simplicity of present programmable logic devices allow designers to implement complex functions without having to compromise with the limitations of standard TTL SSI/MSI devices. Many design changes are easily resolved by merely re-programming devices; the alternative may be extensive changes to a wirewrapped or PC board.

Second-generation devices extend these present benefits even further. Allowing the designer to program the architecture of a device increases his ability to implement even more complex functions. In addition, gate equiva- lency has been increased, allowing more logic functions to be implemented on a single device. But this extended ability does not hinder easy usage of the device. These devices continue to allow for simple design changes as do present-generation devices. A look into some of these second-generation devices will uncover what extended capabilities are available to the designer and how they are beneficial.

AmPAL22V1Ø : "Family of One"

The AmPAL22V1Ø is a 24-pin, second-generation fuse-programmable logic device. This single device encompasses both sequential and combin-atorial logic while maintaining a highly flexible, user-definable archi-tecture. The flexibility of the device's architecture is largely due to its output macrocell structure. There are a total of ten output macrocells in the device. An output logic macrocell, as shown in Figure 1, have one of four possible output combinations: sequential/active high, sequential/active low, combinatorial/ active low or combinatorial/active high. In addition, individual output enables are used to control the I/O terminals to behave as output pins, input pins, or bidirectional ports. The availability of such alternatives allows the designer to modify the architecture and also make the most efficient use of the device for his particular application.

Macrocell Architecture

The macrocell (Figure 2) contains three main logic blocks: a rising-edge-triggered D-type flip-flop with asynchronous reset and synchronous preset inputs, a four-to-one output-path selection mux, and a two-to-one-feedback path selection mux. Two fuses are imbedded in each macrocell to control the four possible output configurations: Rn selects sequential or combinatorial operation; Pn selects active low or active high.

Table 1 outlines the possible states of the fuses --- Rn & Pn --- and lists the resulting output configurations for each case. When both fuses are intact the output is sequential/active low. Programming Pn only configures the output to be sequential/active high. Alternatively, programming Rn only will yield a combinatorial/active low output. Blowing both fuses (Rn & Pn) will yield combinatorial/active high output. When the output is programmed in the combinatorial configuration, the programmable output enable can be used to alter the output

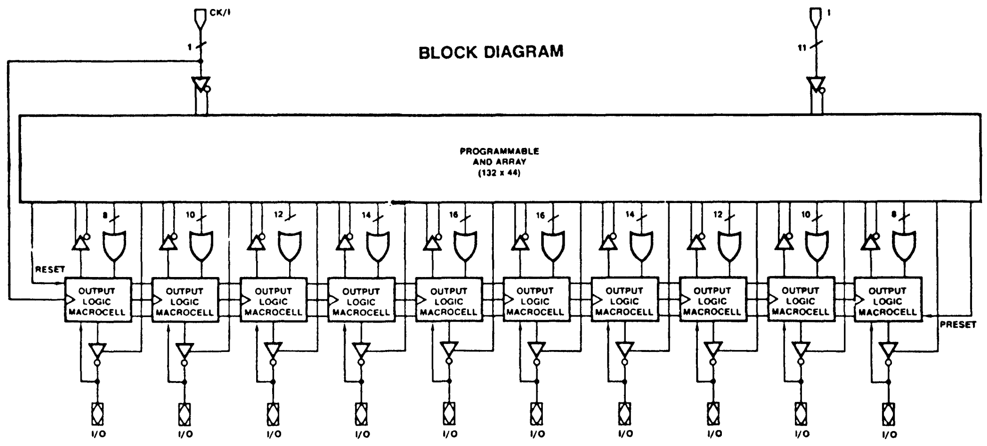


Figure 1

pin to be either a dedicated input or a dynamically controlled I/O port. Once selected as a dedicated input, the pin's output function is sacrificed. This capability is extremely valuable when partitioning functions in a device. The limitation is no longer the number of input pins, but the total number of device pins. As a dynamically controlled I/O, the pin is enabled to serve both functions as needed for the particular application.

Basic Architecture

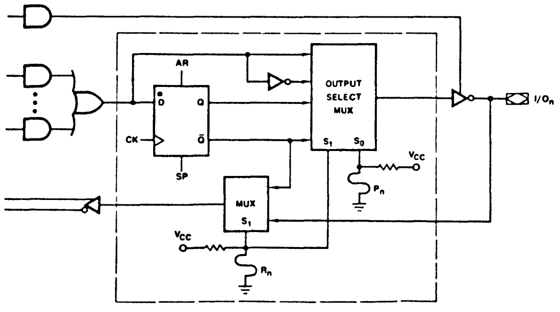
Excluding the extended capabilities, the basic architecture of the AmPAL22V1Ø makes it comparable to other PAL devices. Designers continue to implement logic functions on the device by simply programming the fuse links in the programmable AND-gate array. Incorporating the added flexibility of programming the device's architecture via macrocells increases the design freedom by a substantial amount. However, this feature does not require extra labor or time of the designer.

Rn & Pn Fuse States

Rn	Pn	OUTPUT CONFIGURATION
Ø	Ø	Registered/Active Low
Ø	1	Registered/Active High
1	Ø	Combinatorial/Active Low
1	1	Combinatorial/Active High

Ø = Unblown Fuse
1 = Blown Fuse

TABLE 1



OUTPUT LOGIC MACROCELL

Figure 2

As shown in the block diagram in Figure 1, the device has up to 22 inputs and 1Ø outputs available. There are twelve dedicated inputs and ten with bidirectional (I/O) capability, allowing a maximum of 22 inputs. Of the twelve dedicated inputs, one exists as a shared clock input. Each input has its true and complement version directly connected to a fuse programmable AND-gate. By selectively programming the fuses, the AND gates may be either "connected" to only the true input (by blowing the complement fuses), to only the complement input (by blowing the true fuses), or to neither type of input (by blowing both fuses), thus establishing a logical "don't care." When both the true and complement fuses of a product term are left intact a logical false results on the output of the AND gate. With all fuses blown the output of the gate will assume a logical true state.

Each AND-gate has 44 inputs, resulting in a total of 132 AND-gates in the entire array. From the total, 12Ø AND-gates contribute to the logical product terms, 1Ø are assigned to control each of the three-state outputs, and the remaining 2 activate synchronous register presets and asynchronous register resets.

The array of 12Ø logical product terms are variably distributed. The distribution of product terms range from a minimum of 8 product terms (AND-gates) to a maximum of 16 per output. Note from Figure 1 that the variable distribution of product terms of the upper and lower five group of outputs is symmetrical. The purpose of incorporating the variation of AND-gates in this fashion is its usefulness when implementing counting and complex state machines. A simple example in the next section will demonstrate the versatility of this second-generation device versus present-generation PAL devices.

Typical Application

A registered barrel shifter will be used as an example to demonstrate the versatility of the AmPAL22V1Ø. Most data-processing systems require some form of data-shifting or rotating function. The barrel shifter can provide this elementary function, which may be applicable to such diverse cases as floating point arithmetic and string manipulation.

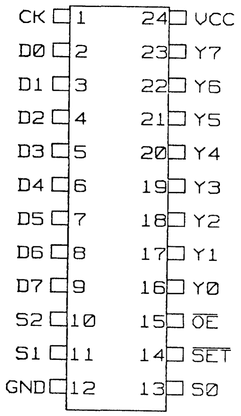
The design of a typical 8-bit registered barrel shifter requires at least 8 inputs, 8 registered outputs, and 3 control lines to implement its functions (see Table 2).

Function Definition Outline

CNTRL INPUTS			INPUT TO OUTPUT MAPPING							
S_2	S_1	$S_Ø$	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	$Y_Ø$
Ø	Ø	Ø	D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_Ø$
Ø	Ø	1	D_6	D_5	D_4	D_3	D_2	D_1	$D_Ø$	D_7
Ø	1	Ø	D_5	D_4	D_3	D_2	D_1	$D_Ø$	D_7	D_6
Ø	1	1	D_4	D_3	D_2	D_1	$D_Ø$	D_7	D_6	D_5
1	Ø	Ø	D_3	D_2	D_1	$D_Ø$	D_7	D_6	D_5	D_4
1	Ø	1	D_2	D_1	$D_Ø$	D_7	D_6	D_5	D_4	D_3
1	1	Ø	D_1	$D_Ø$	D_7	D_6	D_5	D_4	D_3	D_2
1	1	1	$D_Ø$	D_7	D_6	D_5	D_4	D_3	D_2	D_1

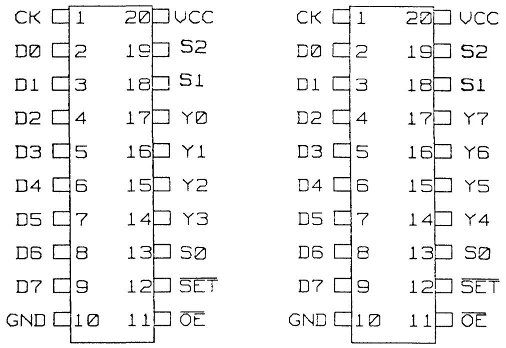
TABLE 2

Figure 3 shows the registered barrel shifter implemented using a single AmPAL22V1Ø. The functions implemented are outlined in Table 2. Note that this single chip easily accommodates the entire function defined in the outline. Suppose the same functions were to be implemented using present-generation programmable logic devices. Two of these devices (AmPAL16R4s) would be required. The question may arise as to why a single AmPAL16R8 could not used. Although this device has 8 registered data outputs, there are only 8 inputs available, which is not sufficient to implement the functions outlined. An alternative solution is to divide the functions into two groups and use two AmPAL16R4s, as shown in Figure 4. This is still a vast improvement over the MSI solutions, which would require 5 chips (4 Am25S1Øs for the barrel shifter, and an Am25S374 octal three-state register).



BARREL SHIFTER IMPLEMENTED WITH SINGLE AmPAL22V10

Figure 3



BARREL SHIFTER IMPLEMENTED WITH TWO AmPAL16R4s

Figure 4

AmPAL18P8: Second Generation 20-pin Device

The AmPAL18P8 is a 20-pin second-generation fuse-programmable device designed to supersede a series of existing 20-pin combinatorial devices. This single device is a functional replacement for all 13 combinatorial devices shown in the tree structure in Table 3. The architecture of the device has been functionally enhanced with such features as increased I/O flexibility, increased logic power, and performance options over present-generation devices. This device extends the PAL* (AND-OR structure) concept, and by incorporating the enhanced capabilities allows maximum user flexibility.

The number of possible inputs has been increased to 18, ten of which are dedicated and eight of which have I/O capability. When any of the latter eight inputs are used as outputs, the user controls the enable of the output as well as its polarity. An increase in logical product terms allows up to 8 terms per output - a maximum of 64 logical product terms are available per device.

The programmability of the enable and polarity of each output increases the flexibility of the device. The availability of up to 18 inputs with 8 logical product terms per output allows for better device utilization in complex applications.

Basic Architecture

The basic architecture of the AmPAL18P8 is shown in Figure 5. The device consists of 18 inputs and 8 available outputs. There are ten dedicated inputs and eight with bidirectional I/O capability allowing a maximum of 18 inputs.

The programmable AND architecture of the AmPAL18P8 is identical to the one described in the AmPAL22V10 section. Every input has its true and complement form directly connected to a fuse-programmable AND array. By selectively programming fuse links, logic functions are implemented on the device.

* PAL is a registered trademark of Monolithic Memories,Inc

The output architecture consists of a total of nine programmable AND gates -- eight are used to define the logic functions and one controls the enable of the inverting output buffer. The eight logical AND gates yield 8 product terms per AND-OR structure. Each of the eight OR gates have the outputs of eight AND gates directly connected. Eight 2-input exclusive-OR gates exist with the output of a fixed-OR gate and a user-programmable polarity fuse (Pn) as their inputs. The polarity fuse allows the user to program the polarity of each input. A blown (unconnected) fuse indicates an active HIGH output and an intact (connected) fuse, active LOW. The output of the exclusive-OR gate is directly connected to the inverting output buffer that is controlled via a programmable enable product term.

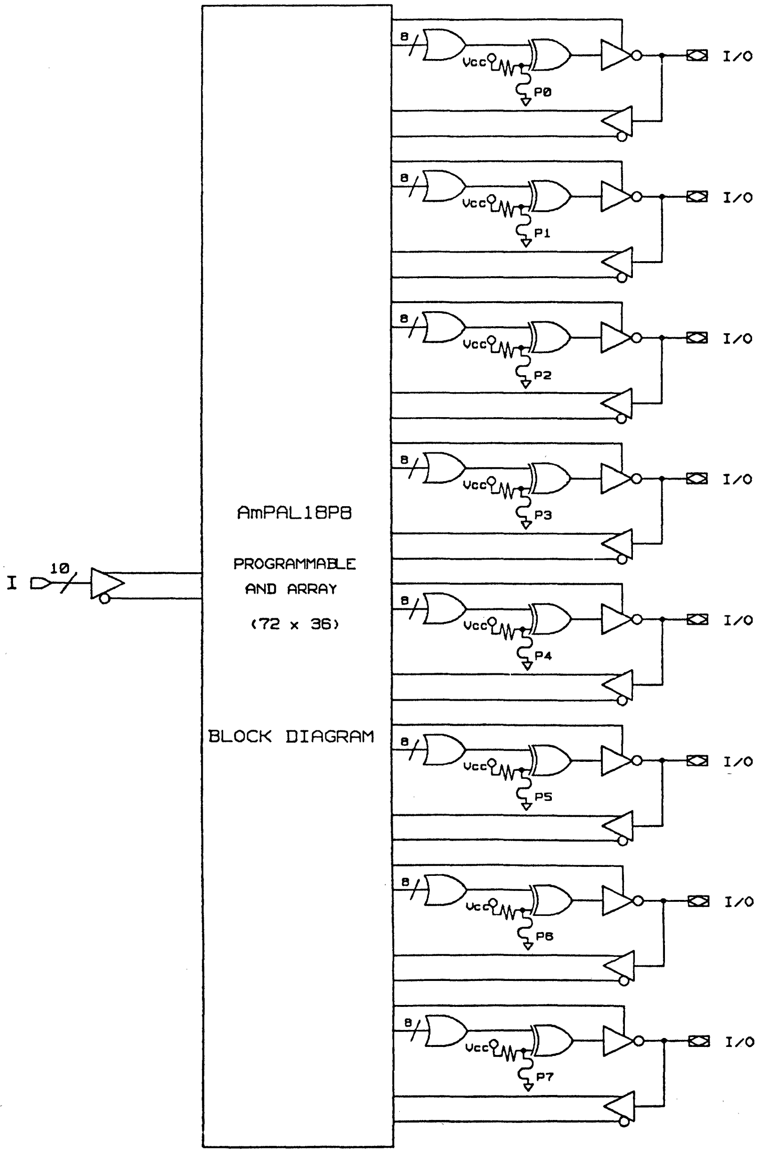


Figure 5

AmPAL18P8 Hierarchy

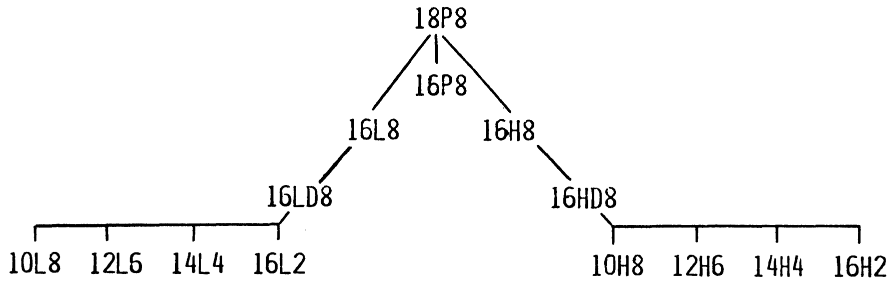


Figure 3

The programmability of the enable and polarity of each output increases the flexibility of the device, and the availability of 18 inputs with 8 product terms available per output allows for better device utilization in complex applications.

APPLICATIONS

Because the AmPAL18P8 is a superset of a series of combinatorial 2Ø-pin devices (16P8, 16L8, 16H8, 16LD8, 16HD8, 1ØL8, 12L6, 14L4, 16L2, 1ØH8, 12H6, 14H4, and 16H2), this single device is not only functionally compatible but also pin-to-pin compatible with this entire series of existing devices. The benefits of these two features are obvious. Functionally, for example, if a design were implemented in an AmPAL16L8, the AmPAL18P8 could be programmed to implement those exact logic functions and also accommodate for any additional logic changes. This is possible because (1) the number of product terms per output has been increased and (2) pins 12 & 19 have been modified to have bidirectional (I/O) capability. Pin compatibility permits flexibility within a single device and also helps to reduce parts inventory. Thus, the device can be configured to be a direct replacement for any one of those devices listed above.

CONCLUSION

This paper has presented several second-generation programmable logic devices and described the benefits associated with them. Classification as "second-generation" identifies the architecture of these devices to be designer-definable beyond the extent of present programmable logic devices. The AmPAL22V1Ø was presented as a dynamic device in which the output architecture is fully defined by the designer via output macrocells. The benefits and design applicability were discussed. Another such device was the AmPAL18P8. This single device was presented as a superset to a series of 2Ø-pin devices, and its benefits and design applicability were discussed.

Méthode de test et de conception de diagnostic

Faisant suite en quelque sorte à l'article paru dans « minis et micros » numéro 202 et traitant des registres espions, cette étude présente un modèle architectural possible d'un système entièrement testable incorporant des registres de diagnostic et des Prom à registre de diagnostic, et explique les choix qui ont conduit à l'adoption de ce modèle. La testabilité de chacun des blocs fonctionnels constituant le système est étudiée en détail.

A mesure que l'emploi des ordinateurs continue de se banaliser, l'intérêt se porte beaucoup plus sur les aspects test et fiabilité des systèmes. Les constructeurs et les utilisateurs se sont rendu compte que leur coût d'entretien pendant un cycle complet de fonctionnement dépassait de beaucoup leur prix d'achat initial.

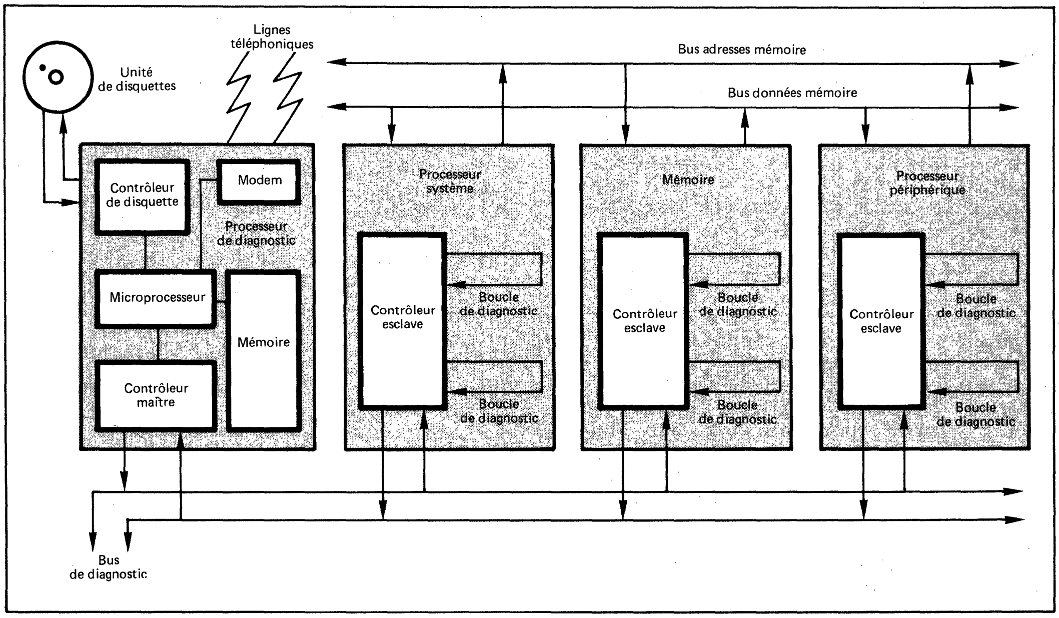
Les utilisateurs de systèmes prennent maintenant beaucoup plus en considération des spécifications telles que l'intervalle moyen entre les pannes (MTBF), le prix des réparations, leur durée moyenne (MTTR) et les niveaux de confiance du diagnostic. La méthode Serial Shadow Register ou SSR (voir « minis et micros » numéro 202) est un exemple de réalisation simple permettant d'augmenter la fiabilité et les possibilités de test d'un système. Cette méthode fournit au concepteur une technique systématique de détection des défaillances du matériel en assurant l'accès à des états internes cachés.

Dans la conception d'un système que l'on peut entièrement diagnostiquer, l'aspect le plus important consiste à obtenir un niveau maximal de possibilités d'observation et de contrôle avec un minimum de matériel supplémentaire. La **figure 1** décrit un modèle architectural possible d'un système SSR rapide dont nous allons montrer les possibilités de diagnostic.

Un processeur de diagnostic commande la réalisation de tests « carte par carte » selon un mode maître-esclave. Le processeur utilise les données et le code de diagnostic disponibles à partir d'une disquette ou d'un modem s'ils se trouvent à un endroit éloigné, et manipule un contrôleur de diagnostic maître. Ce dernier transmet les informations en série aux contrôleurs esclaves situés sur chaque carte du système, qui réalisent les tests demandés en commandant les boucles de diagnostic appropriées. Les résultats sont retransmis au processeur qui les évalue.

L'isolement et le test de chaque carte du processeur central, de la mémoire et du processeur périphérique font partie de la stratégie « diviser pour régner » utilisée tout au long de cet exemple, en association avec la méthode SSR. Le processeur central est divisé en deux blocs fonction- ▶

Fig. 1 - Dans ce modèle architectural utilisant des SSR (serial shadow register) un processeur de diagnostic maître commande les tests de chaque carte.



▶ nels plus petits, l'unité centrale proprement dite et le séquenceur/mémoire de commande gérant la mémoire et les signaux d'horloge.

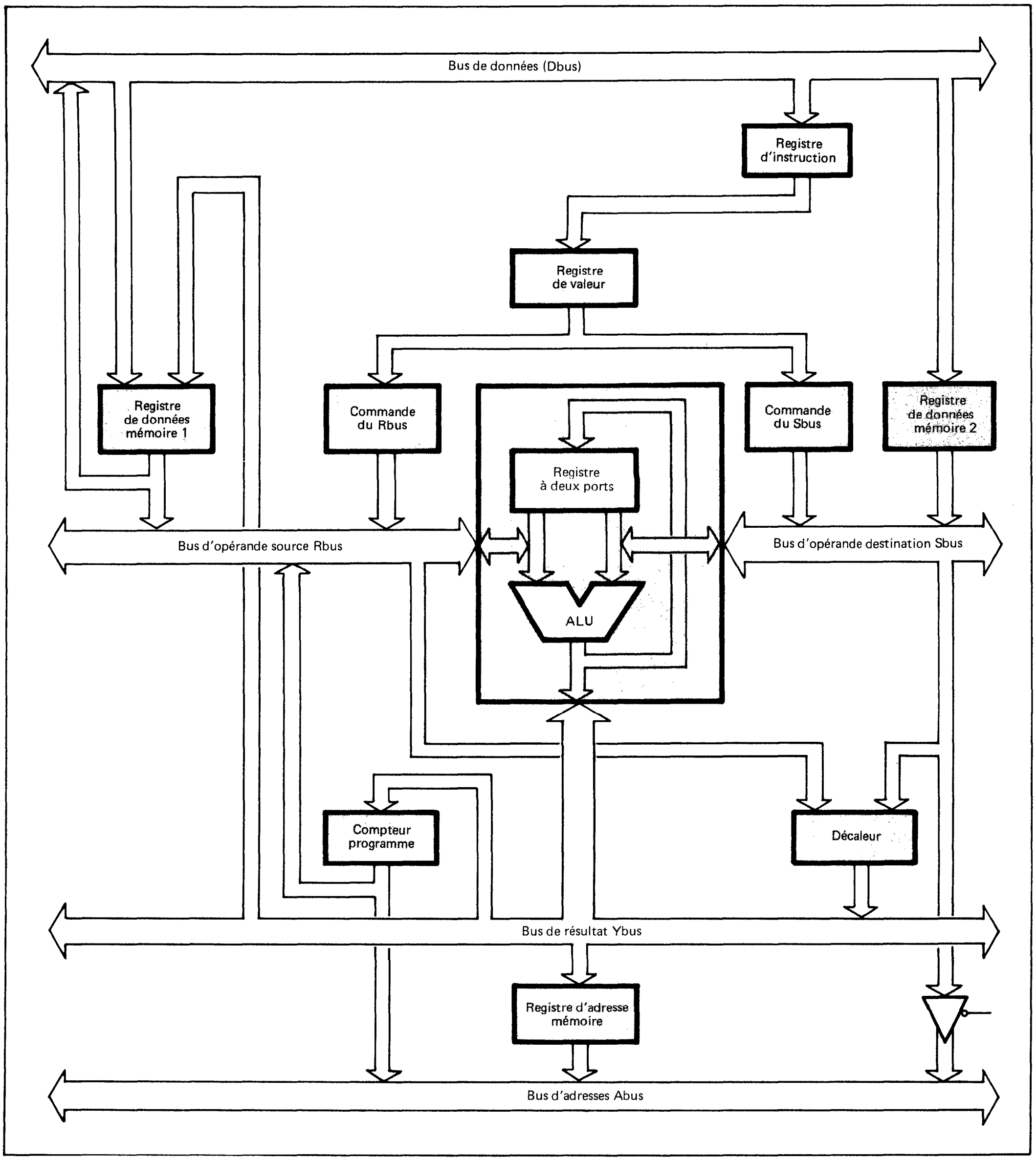
Architecture de l'unité centrale

Elle repose sur un processeur à deux adresses et à registres et se caractérise par des bus multiples et des blocs fonctionnels possédant plusieurs entrées/sorties. Ce type d'organisation **(fig. 2)** facilite les opérations en parallèle et augmente donc les performances du système. Les cinq bus principaux sont : le bus de données (Dbus), le bus d'adresse (Abus), le bus d'opérande source (Rbus), le bus d'opérande destination (Sbus), et le bus de résultats (Ybus). Les dix blocs fonctionnels sont : le bloc de registre et l'unité arithmétique (Am 29203), le compteur de programme (PC), le registre d'instruction (IR), le registre de valeur (VR), le décaleur combinatoire, la commande du Rbus, la commande du Sbus, le registre d'adresse mémoire (MAR) et les deux registres de données mémoire (MDR1 et MDR2).

Les blocs fonctionnels

L'unité arithmétique et le bloc registre sont composés de microprocesseurs bipolaires en tranches Am 29203. Le bloc registre interne de l'Am 29203 est à double port. On accède au port A par le Rbus et au port B par le Sbus. Les résultats des

Fig. 2 - L'unité centrale est composée de cinq bus et de blocs fonctionnels organisés en parallèle.



opérations de l'ALU se trouvent sur le Ybus.

Le décaleur combinatoire — bloc entièrement combinatoire composé d'Am PAL16H8A — peut décaler un nombre arbitraire de bits en un seul cycle. Ses sources d'entrée sont le Rbus et le Sbus et les résultats peuvent apparaître sur le Ybus.

Le registre d'instruction (IR) bénéficie de la logique interne des Am PAL16H8A et l'utilise pour le prédécodage rapide des instructions, en réduisant le nombre de bits du IR demandés par le séquenceur/mémoire de commande. Les autres bits d'information du IR que le séquenceur/mémoire de commande n'utilise pas sont employés par le registre de valeur.

Ce dernier contient les informations relatives aux instructions comme l'adresse de branchement conditionnel, les valeurs immédiates de données et/ou les adresses du bloc registre. Le registre de valeur est un élément essentiel dans la mise en place d'une architecture en pipeline double.

On utilise les blocs de commande du Rbus et du Sbus pour l'extension signée des opérandes courts immédiats, la génération de déplacements sur les branchements conditionnels et la génération de constantes. Les entrées de ces deux blocs proviennent du registre de valeur.

Le compteur de programme consiste essentiellement en un registre à incrémentation qui peut être chargé en parallèle à partir du Ybus. Pour les opérations de recherche en mémoire c'est une source du Abus et pour les calculs de branchement relatif c'est une source du Rbus.

Les adresses des opérandes source et destination sont stockées dans le registre d'adresse mémoire et proviennent toutes du Ybus.

Les registres MDR1 et MDR2 stockent les données provenant de la mémoire. MDR1 sert à stocker les opérandes source et MDR2 stocke les opérandes de destination. Outre cette fonction, MDR1 reçoit les résultats des opérations de l'ALU à destination de la mémoire et les place sur le Dbus.

Diagnostic SSR appliqué à l'unité centrale

Une fois cette première étape de conception terminée, l'intégration du diagnostic SSR à l'unité centrale peut commencer. En fait, les deux étapes devraient être menées de front. Pour plus de clarté, nous les avons considérées séparément. Le travail du concepteur de système consiste à placer des matériels de diagnostic dans les sous-ensembles essentiels du système, de manière à pouvoir contrôler et observer tous les chemins principaux et les éléments logiques. Pour ce faire, il est évident que chacun des cinq bus principaux doit être accessible, ce qui est facilité par la méthode SSR. En utilisant des registres pipeline Am 29818 pour les registres MDR1, MDR2 et le registre d'adresse mémoire, on peut réaliser des essais complets de tous les bus.

Le test des bus

Cependant, avant d'exécuter un diagnostic sur l'un des bus du système, il faut tester les boucles de diagnostic ; tout d'abord, on décale un seul 1 sur chaque boucle, en vérifiant les erreurs, puis on répète la même opération avec un seul 0. Une fois vérifié le bon fonctionnement des boucles, on peut passer à la vérification du système.

Les diagnostics de bus permettent de détecter des blocages sur 1 ou 0 et les court-circuits entre lignes. On peut détecter les défaillances en blocage sur 1 en utilisant une configuration de tous les 0, toute configuration différente reçue indiquant alors une erreur. De même, des défaillances en blocage sur 0 pourraient être détectées en utilisant une configuration de 1.

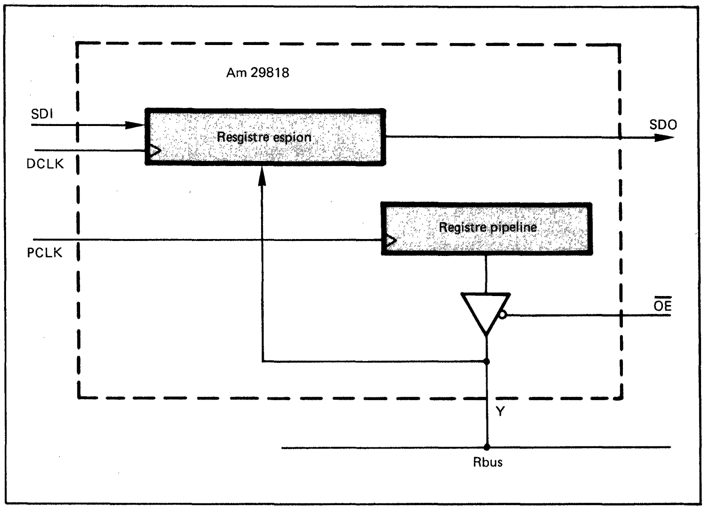
On peut utiliser des séquences de test à « 0 ou 1 baladeur » pour détecter des court-circuits sur un bus. La séquence « en 0 baladeur » consiste en une série de configurations binaires, chacune contenant un seul 0 placé à différents endroits. Dans la première configuration, le 0 peut se trouver en LSB (Least significant bit), dans la deuxième juste avant le LSB et ainsi de suite. De même, la séquence en 1 baladeur consiste à placer un seul 1 à des endroits différents. Avec la configuration de tests en 0 baladeur, un résultat comportant plus d'un 0 indique la présence d'un court-circuit. Deux 1 ou plus indiqueraient la même chose pour la séquence en 1 baladeur. Si l'on connaît les bits du bus adjacents, le code diagnostic pourra déterminer les lignes en court-circuit.

Grâce à la conception de l'Am 29818, un seul registre est souvent suffisant pour détecter toutes ces défaillances. Par exemple, MDR1 peut être utilisé seul afin de vérifier chaque défaillance possible du Rbus. Tout d'abord, on décale une configuration à l'intérieur du registre espion (Shadow Register) de MDR1, puis on la charge dans le registre pipeline. Celui-ci est validé sur le Rbus en même temps que les Am 29818 sont positionnés pour charger l'information des broches Y dans le registre espion. Puisque les broches de sortie Y sont connectées au Rbus, on échantillonne en fait l'information présente sur le Rbus (voir **fig 3**). Un changement de la configuration indiquera une défaillance particulière.

Le test du Ybus

Chaque bus du système — sauf le Ybus — peut être testé de cette façon. Le Ybus doit être traité différemment car aucun bloc fontionnel utilisant des registres de diagnostic en pipeline n'alimente ce bus. Là encore, l'architecture de l'Am 29818 permet de résoudre ce problème. On peut le positionner dans un mode où le contenu du registre espion soit présent sur les broches bidirectionnelles D. Bien que ce mode ne puisse ▶

Fig. 3 - L'architecture de l'Am 29818, permet le test du Rbus.



▶ fournir que 4 mA, la quantité de charges sur le Ybus ne pose pas de problème.

On peut ainsi utiliser le registre d'adresse mémoire pour inscrire des données de diagnostic sur le Ybus au moyen de ses broches D (voir **fig. 4**). C'est le registre MDR1 qui lit les résultats du Ybus. Il faut cependant remarquer que, puisque MDR1 peut recevoir des données provenant de deux chemins différents, il faut le forcer pour accepter les données du Ybus, ce qui est possible en utilisant des AM 29818 pour le registre pipeline de micro-instruction du séquenceur et en y décalant les bits de commande nécessaires (le registre pipeline de micro-instruction est un endroit propice pour utiliser les Am 29818 puisqu'il renferme tous les bits de commande de chaque état du processeur central).

Etant donné que les données reçues par MDR1 doivent transiter à travers la logique, une défaillance détectée sur le Ybus ne signifie pas obligatoirement qu'il existe un problème sur ce bus. Cela peut être dû à la logique elle-même. Afin d'établir la vraie cause, il faut donc effectuer des essais plus poussés, et essayer un autre chemin utilisant le Ybus et non MDR1. Un chemin possible serait de partir de MDR2 pour arriver au registre adresse mémoire (MAR) en passant par l'ALU. Mais avant de l'utiliser, il convient de vérifier le bon fonctionnement de l'ALU. Ensuite, on peut y faire transiter une configuration et la charger dans le MAR. Si aucune défaillance n'est détectée, ce n'est pas le bus qui est responsable, mais peut-être la logique située en amont de MDR1.

L'intégration du diagnostic SSR n'entraîne pas de modification d'architecture ; seul du matériel supplémentaire est nécessaire. Comme MDR1 doit fournir des données à deux bus, il est nécessaire d'installer un deuxième registre si l'on veut maintenir l'efficacité de l'Am 29818. L'un de ces registres alimentera le Dbus et l'autre le Rbus. Il n'y aura aucun changement dans le nombre de boîtiers puisque les Am 29818 supplémentaires remplaceront les émetteurs à trois états. Le registre supplémentaire n'est absolument pas nécessaire à l'exploitation du processeur central, mais il accroît les possibilités de test du système. Or, comme c'est l'un des buts essentiels de la conception, l'avantage retiré est sûrement beaucoup plus à prendre en considération que son prix.

Il faut aussi savoir qu'en utilisant le MAR pour alimenter le Ybus en données de diagnostic, tel qu'on l'a décrit plus haut, un problème peut surgir selon la boucle de diagnostic où il réside. Comme tous les Am 29818 d'une boucle sont commandés par les mêmes signaux DCLK, Mode et SDI, chaque dispositif de la boucle MAR émettra des informations en même temps que lui. Or, si on ne prend pas de précaution, cela peut entraîner de graves conflits entre les bus. Pour les éviter, le moyen le plus simple et le plus efficace est d'isoler le MAR dans une boucle séparée. Ce dernier doit effectivement se trouver dans une boucle différente de celle des deux MDR ; quant à savoir si les MDR doivent se trouver sur la même boucle, cela dépend du concepteur. Pour obtenir des performances et une souplesse optimales, il est souhaitable de séparer les boucles, mais le contrôle s'en trouve alors compliqué. Néanmoins, ce problème ne nous intéresse pas ici, et nous l'examinerons plus tard.

Le test de L'ALU

Afin de procéder à des essais exhaustifs de l'ALU et du bloc registre, on doit pouvoir contrôler les signaux de commande envoyés à l'Am 29203. Comme pour MDR1, cela peut être réalisé en utilisant des Am 29818 pour le registre pipeline de

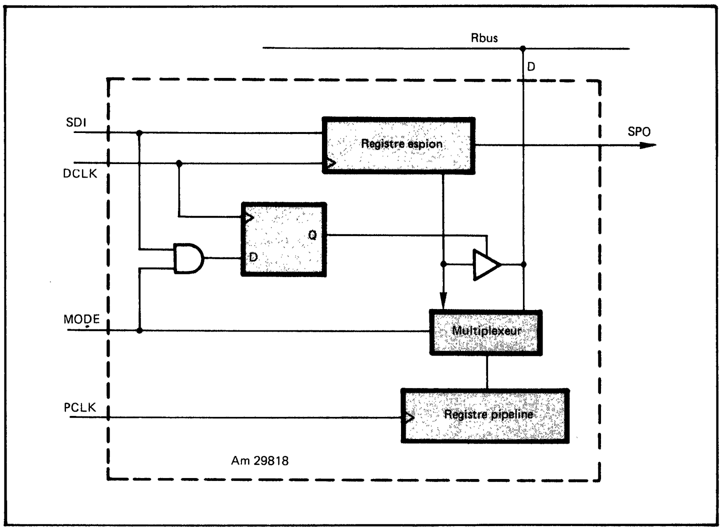
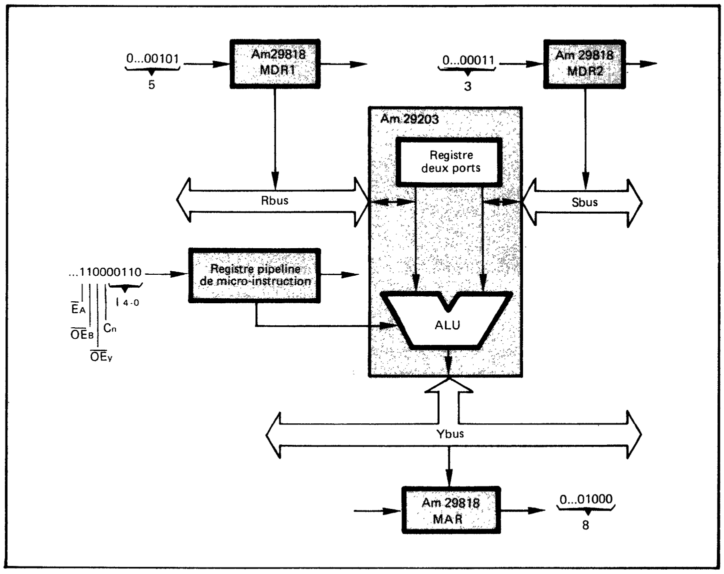


Fig. 4 - Un mode spécial de l'AM 29818, utilisant les broches D, permet de tester l'Ybus.

Fig. 5 - Le test de l'unité arithmétique et logique se fait par l'intermédiaire des registres MDR1, MDR2 et du registre d'adresse mémoire.



micro-instruction, ce qui permettra un contrôle complet de chaque bloc fonctionnel de l'unité centrale. On peut exécuter les fonctions de l'ALU en chargeant tout d'abord les bits de commande nécessaires dans le registre pipeline de micro-instruction, puis en inscrivant les données sur le Sbus au moyen du registre espion de MDR2. Les données du Rbus sont générées de la même façon en utilisant MDR1. L'ALU exécute l'opération demandée et le résultat peut être chargé dans le MAR. Les données sont transférées du registre pipeline du MAR vers son registre espion et sont ensuite envoyées à l'extérieur afin d'être analysées.

Le premier test applicable à un bloc logique combinatoire est celui des liaisons logiques. En effet, il convient de s'assurer que tous les blocs logiques sont correctement connectés entre eux. On peut le vérifier en regardant si les données transférées par l'Am 29203 du Sbus ou du Rbus vers le Ybus restent inchangées. Une fois cela fait, les essais de fonctionnement proprement dit peuvent commencer.

La **figure 5** représente une addition du contenu du Sbus avec le contenu du Rbus. Tout d'abord on charge les bits de commande nécessaires dans le registre espion du registre pipeline de micro-instruction de l'Am 29203. Les lignes d'instruction destinées à l'ALU, (I_0 à I_4) sont positionnées à 00110 (instruction d'addition) et la retenue C_n est positionnée à zéro. Les entrées de commande E_A et $\overline{OE}_B$ sont toutes les deux positionnées à 1, ce qui valide le Sbus et le Rbus comme opérandes source. $\overline{OE}_Y$ est positionnée à zéro, validant la sortie de l'ALU vers le Ybus. Les données voulues sont chargées à l'intérieur des registres espion de MDR1 et MDR2, 5 et 3 dans notre exemple. Les registres pipeline de micro-instruction, MDR1 et MDR2 sont alors chargés par leurs registres espions respectifs et l'opération est exécutée. Les résultats apparaissent sur le Ybus et sont chargés dans le registre pipeline du MAR puis dans son registre espion, d'où ils sont envoyés à l'extérieur pour être analysés.

Toutes les fonctions de l'ALU peuvent être testées de cette manière. Ayant testé l'ALU de l'Am 29203, on peut examiner le bloc de registre interne, en utilisant MDR1 ou MDR2 comme sources de données et en faisant transiter les données, de l'ALU vers le bloc registre. On peut alors lire les données présentes aux ports A et B de MDR1 et MDR2 respectivement. En utilisant ces procédés, l'Am 29203 peut être testé sans difficultés.

On peut appliquer au décaleur combinatoire le même procédé de diagnostic utilisé pour l'ALU et le bloc registre. On utilise MDR1 et MDR2 pour contrôler les données fournies au décaleur combinatoire et on se sert du MAR pour observer les résultats. Dans ce cas, le test des liaisons logiques est effectué en décalant les positions de zéros des données.

Le test des autres blocs fonctionnels

A ce stade, il reste à tester le registre d'instruction, le registre de valeur, les blocs de commande du Sbus et du Rbus et le compteur de programme. En remplaçant le registre de valeur par des Am 29818, le concepteur aura la totale maîtrise des tests concernant le registre de valeur et les blocs de contrôle du Sbus et Rbus. Pour ces blocs fonctionnels, on peut appliquer le même procédé utilisé pour l'ALU et le décaleur combinatoire.

Le PC et le IR restent donc les derniers blocs de l'unité centrale à tester. Rappelons qu'un réseau logique programmable (PAL) à registres a été utilisé pour composer ces deux blocs. Puisque le VR est composé d'Am 29818, IR peut rester tel quel. Les tests seront effectués en transmettant des données au Dbus à l'aide de MDR1 et en chargeant ces données dans les registres de l'IR par l'intermédiaire des PAL. Ces données seront ensuite chargées dans le VR et envoyées à l'extérieur pour être examinées.

Néanmoins, on ne pourra pas faire immédiatement la distinction entre un problème d'interconnexion entre l'IR et le VR, et une défaillance interne de l'IR. Du point de vue de l'amélioration des possibilités totales de tests, une meilleure solution consisterait à isoler le registre du PAL. On pourrait employer des Am PALI6H8 devant les Am 29818 afin d'assurer les mêmes fonctions et permettre un diagnostic plus complet (la méthode SSR pourrait être ensuite incorporée aux composants des réseaux logiques programmables). A ce stade on peut réaliser une vérification complète de l'IR à l'aide du procédé normal.

Le même procédé peut être appliqué au compteur de programme. Dans ce cas, le MAR fournit les possibilités de contrôle et le registre espion du PC fournit les possibilités d'observation.

L'intégration du diagnostic SSR dans l'unité centrale a été assez simple et directe. Aucun changement d'architecture et peu de modification de matériel ont été nécessaires pour réaliser les essais sur cette partie du système. Cette tendance continuera, les autres parties du système étant renouvelées de manière à inclure le diagnostic SSR.

***

Le second volet de cet article, qui paraîtra dans un prochain numéro de « minis et micros » sera consacré à l'étude et au test de la deuxième partie constitutive de l'unité centrale, le séquenceur/mémoire de commande.

Kevin Ow-Wing

Méthode de test et de conception de diagnostic (2^e partie)

Après nous être attachés dans notre précédent numéro (« minis et micros » numéro 205) à l'architecture et au diagnostic SSR (Serial Shadow Register) appliqués à l'unité centrale, nous allons étudier, dans cette deuxième partie, l'architecture et le test du séquenceur mémoire de commande et des processeurs périphériques.

Rappelons que le processeur central est divisé en deux blocs fonctionnels, l'unité centrale proprement dite et le séquenceur/mémoire de commande. Ce dernier gère la mémoire et les signaux d'horloge.

Architecture du séquenceur/mémoire de commande

La **figure 1** représente un bloc diagramme du séquenceur/mémoire de commande. Les sept blocs fonctionnels sont : la Ram de mémoire de commande, le registre pipeline de micro-instruction, le microséquenceur, le registre d'état, la logique de condition d'exécution, le registre d'état, le registre d'instruction et la commande de l'horloge du système.

L'élément le plus important d'un tel système est la mémoire de microprogramme. Elle contient tous les signaux de commande pour chaque état du processeur central. A cette fin, on utilise des Ram statiques à 45 ns de 4 K × 4 (Am 9168-45 s).

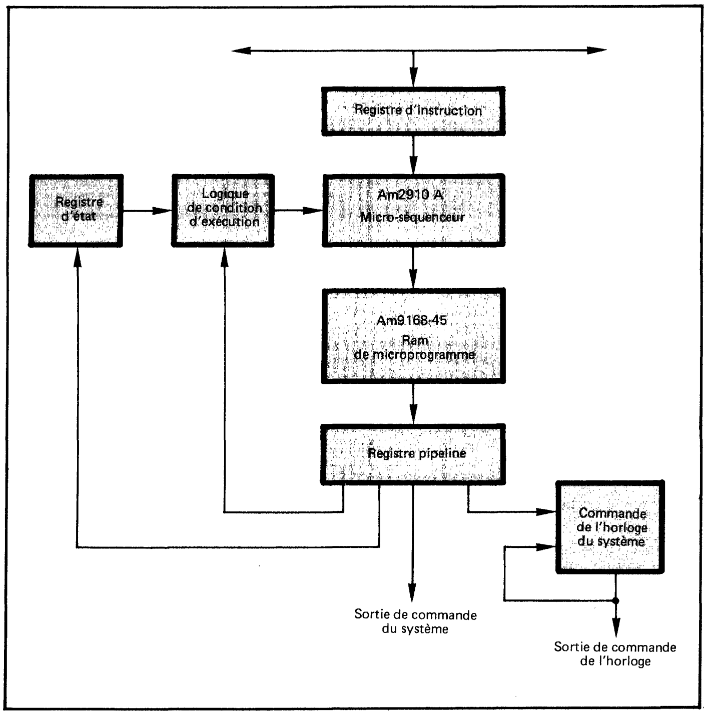
On utilise pour le microséquenceur le circuit Am 2910 A, qui a une capacité d'adressage pouvant atteindre 4 096 mots de microcode avec un temps de cycle de 100 ns.

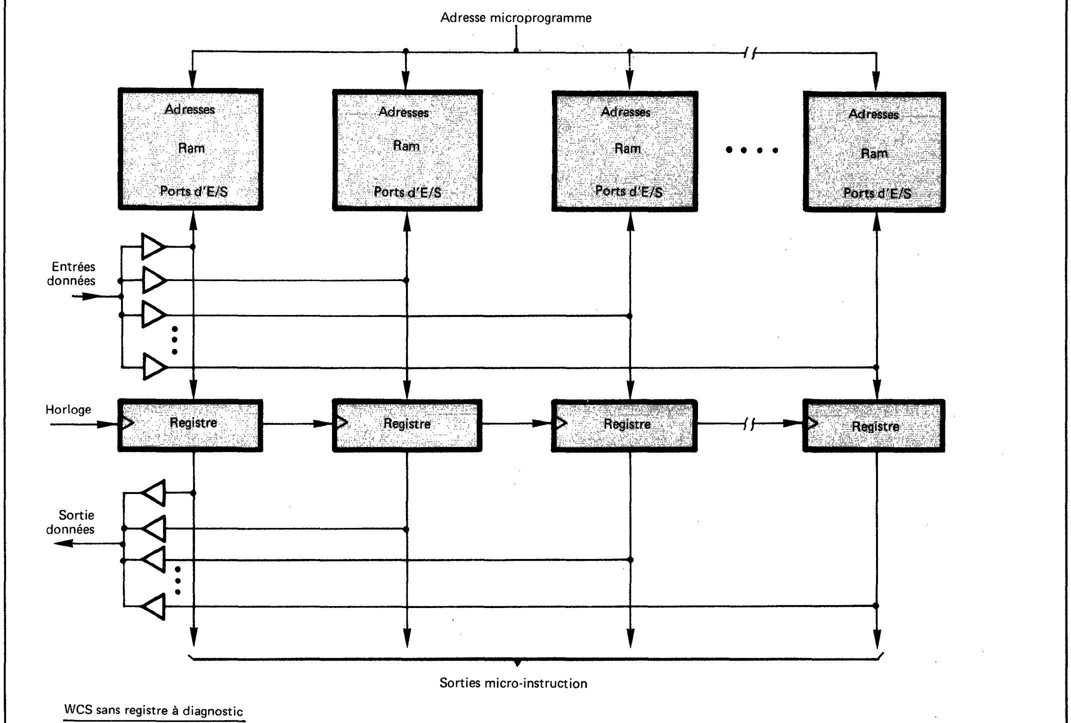
Le registre d'état renferme des informations sur les opérations de l'ALU et du décaleur combinatoire (résultat nul, négatif, etc.). La logique de condition d'exécution utilise les bits d'états comme entrées afin de déterminer l'état des branchements conditionnels. Le microséquenceur utilise à son tour ces informations afin de décider si un microbranchement est nécessaire.

Le bloc de contrôle de l'horloge du système est un ensemble figé, chargé de commander l'horloge et de fournir les signaux de lecture et d'écriture de la mémoire. Il est synchronisé à un débit beaucoup plus rapide que le système afin que l'essai de la fonction « mémoire-prêt » puisse être effectué plus souvent ; il est optimisé afin de confirmer les signaux de lecture ou d'écriture de la mémoire dès que l'adresse mémoire est stable sur le Abus. Cet accroissement des performances prend en compte le fait que l'inscription d'une adresse sur le Abus à partir du PC, du MAR et de MDR2 demande moins de temps qu'à partir du bloc de registre interne de l'Am 29203. Pour réaliser ces opérations, on utilise une Prom à registres.

Afin de permettre un contrôle complet du processeur central, le registre pipeline de micro-instruction est composé d'Am 29818. L'utilisation de ces circuits pour le registre d'état permet également d'observer les bits d'états après les opérations de l'ALU et de

Fig. 1 - Bloc-diagramme du séquenceur/mémoire de commande. L'élément essentiel est la mémoire de microprogramme constituée par des circuits Am 9168.


Registre d'instruction
Registre d'état
Logique de condition d'exécution
Am2910 A Micro-séquenceur
Am9168-45 Ram de microprogramme
Registre pipeline
Commande de l'horloge du système
Sortie de commande du système
Sortie de commande de l'horloge



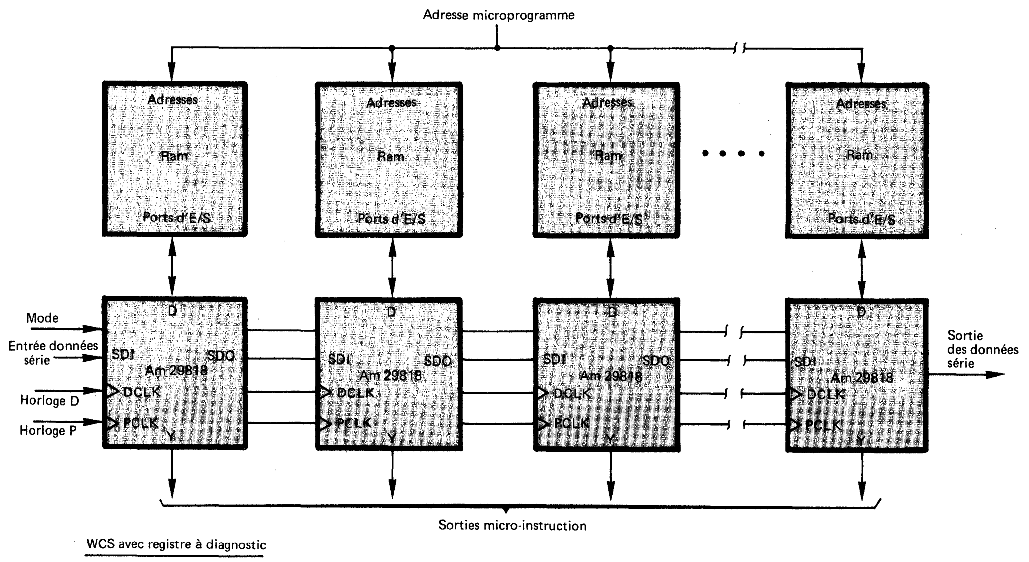


Fig. 2 - L'utilisation de registre pipeline à diagnostic permet de simplifier la conception des mémoires de commande inscriptibles.

contrôler ces bits pendant la réalisation du diagnostic sur le séquenceur/mémoire de commande.

Cela simplifie également la conception de mémoires de commande inscriptibles (WCS pour Writable Control Store). En examinant la **figure 2**, on se rend compte qu'un matériel moins important est nécessaire à l'installation des WCS si on utilise des registres pipeline de diagnostic. Outre la simplification, cette caractéristique assure l'accès nécessaire à la Ram pour effectuer des essais de diagnostic.

Test de la Ram

Pour tester la Ram, il est nécessaire d'effectuer un contrôle à la fois des adresses et des données. L'adresse est acheminée par les Am 29818 du registre d'instruction. Le microséquenceur transfère directement cette adresse à la Ram. Les données de test devant être chargées dans la Ram sont décalées à l'intérieur du registre espion et, à l'aide d'une impulsion d'écriture, sont inscrites par l'intermédiaire des entrées D des Am 29818. Le contenu de la Ram est alors chargé dans le registre espion qui décale les données à l'extérieur afin qu'elles soient évaluées. On peut utiliser de nombreuses configurations pour remplir la Ram afin de tester sa sensibilité.

Néanmoins, cette architecture ne permet pas d'imputer les erreurs détectées au moment du test à la Ram ; en effet l'adresse de la Ram pouvait être mauvaise à la sortie du microséquenceur. Il est donc nécessaire d'avoir un accès direct à l'adresse de la Ram. Dans ce cas, l'addition de deux Am 29818 pour générer l'adresse fournira les possibilités voulues de contrôle et permettra d'adresser la Ram directement, d'éliminer le séquenceur du chemin et d'augmenter les possibilités de test du système.

Test du microséquenceur

L'addition de ces deux circuits aux entrées de l'adresse Ram permet également d'observer les sorties du séquenceur. Le contrôle complet de ce dernier est fourni par le registre pipeline de micro-instruction et le registre d'instruction. Le compteur d'instruction (micro PC) et la pile de neuf mots composant le microséquenceur **(fig. 3)** peuvent être testés de la façon suivante : pour le micro PC, on place d'abord le registre pipeline de micro-instruction de manière qu'il force l'Am 2910 A à choisir ses entrées D comme source de données. Le registre d'instruction est relié à ces entrées et on peut donc l'utiliser pour fournir les configurations de test de diagnostic. La configuration de test est échantillonnée aux sorties du séquenceur en même temps qu'elle est chargée dans le micro PC, ce qui permet de vérifier également les liaisons logiques. Le contenu du registre de micro-instruction est alors modifié afin de faire sortir le contenu du micro PC par le séquenceur.

Si le micro PC fonctionne correctement, la pile de l'Am 2910 A peut être chargée avec les données. On fait sortir ensuite le contenu de la pile par le séquenceur, et on l'échantillonne avec les Am 29818. On peut tester les autres blocs du microséquenceur de la même manière que le micro PC et la pile de neuf mots.

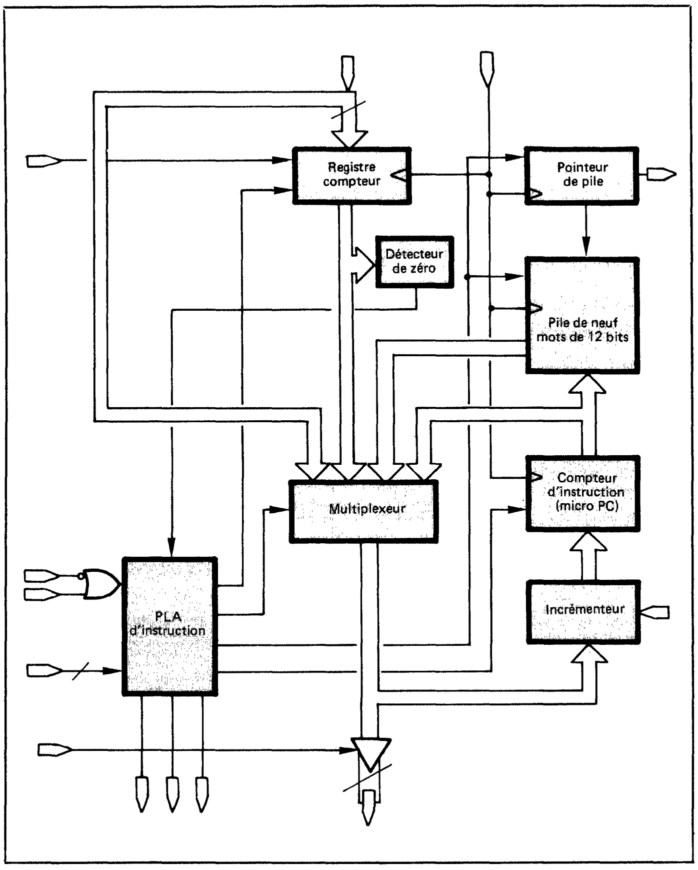


Fig. 3 - Le compteur d'instruction (micro PC), la pile de neuf mots et les autres blocs fonctionnels du micro-séquenceur peuvent également être testés par la méthode SSR.

On peut effectuer la vérification de la logique de condition d'exécution en utilisant le même procédé appliqué aux précédents blocs logiques combinatoires comme le décaleur combinatoire de l'unité centrale. Le registre d'état Am 29818 effectue le contrôle des entrées arrivant à ce bloc logique. Malheureusement, la sortie de cette logique ne peut être observée facilement. On peut résoudre cette difficulté mineure en utilisant, à partir de cette sortie, un chemin qui se dirigerait vers un bit inutilisé du registre Am 29818. Des bits inutilisés sont facilement disponibles si on ajoute deux registres à diagnostic à la sortie à 12 bits du microséquenceur. Même si on ne disposait pas de registre de secours, l'addition d'un seul Am 29818 fonctionnant dans ces conditions en vaudrait la peine. Maintenant qu'il est possible de contrôler et d'observer la logique de condition d'exécution, on peut tester ce bloc complètement en utilisant les techniques précédentes.

Test de la commande d'horloge

Le dernier bloc fonctionnel à vérifier est la commande de l'horloge du système, composée de Prom à registres. S'il existe bien un contrôle des entrées, par contre aucun ne s'exerce sur les variables d'état, de sorte que l'observation des sorties n'est pas facile. On pourrait résoudre ces problèmes en isolant la Prom du registre et en remplaçant ce dernier par un Am 29818 mais il existe une autre solution.

Le contrôle et l'observation désirés peuvent être obtenus beaucoup plus facilement en incorporant les Prom à registres de la famille Am 27S65/75/85 avec fonctions de diagnostic SSR. Tout en permettant le diagnostic, cette solution maintient le niveau voulu d'intégration. Maintenant que la commande de l'horloge est complètement sous contrôle, les essais peuvent commencer.

Afin d'effectuer une vérification complète de l'horloge, on doit pouvoir réaliser toutes les transitions d'états et démontrer que le système peut récupérer tous les états indéfinis. On lui transmet des informations en décalant les signaux nécessaires dans le registre pipeline de micro-instruction. De la même façon, on prépare l'état de l'appareil au moyen du registre espion de l'Am 27S85. Il est important de remarquer que cette technique équivaut à transformer ce réseau séquentiel en un réseau combinatoire, le chargement en série du registre espion étant le même que l'application de vecteurs de tests, ce qui rend possibles tous les tests. Le système est ensuite échantillonné et le nouvel état apparaît dans le registre de la Prom. Ces données sont transférées au registre espion et envoyées à l'extérieur pour être analysées. Ce procédé permet de réaliser un diagnostic complet du dispositif d'horloge ou d'autres applications de ce type.

Il suffit de petites modifications d'architecture et de matériel pour incorporer le diagnostic SSR au processeur central, et on peut maintenant le tester entièrement. Les seuls circuits supplémentaires ont été un deuxième MDR1 et deux boîtiers Am 29818 placés judicieusement à l'intérieur du séquenceur/mémoire de commande. Les techniques décrites pour appliquer le diagnostic SSR se mettent en œuvre facilement et peuvent être utilisées pour une grande variété de réseaux logiques.

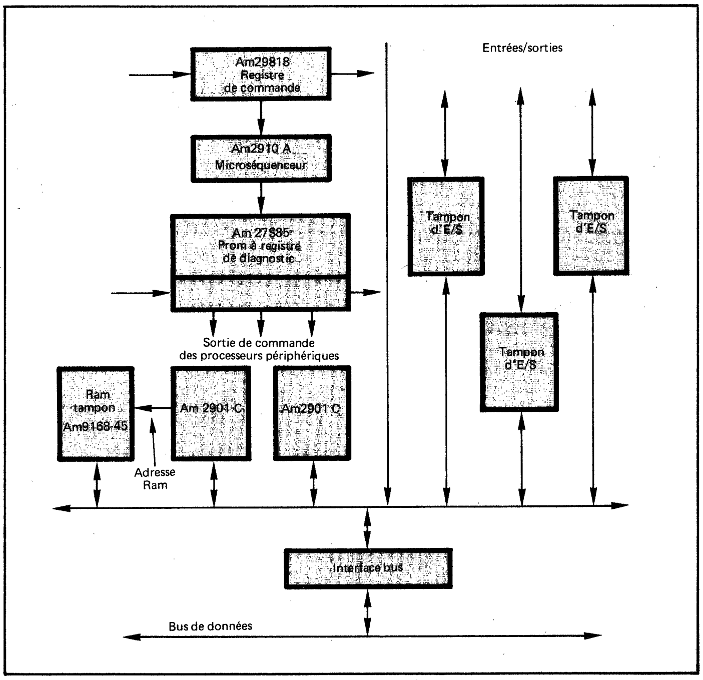


Fig. 4 - Bloc-diagramme d'un processeur périphérique.

Processeurs périphériques

La **figure 4** représente le bloc diagramme d'un processeur périphérique à haute performance utilisant des microprocesseurs bipolaires en tranches, une Ram statique ultra-rapide Am 9168 de 4 K × 4, des Prom ultra-rapides bipolaires à registres Am 27S85 A comportant des fonctions de diagnostic et un microcontrôleur Am 2910 A.

Le registre de commande composé d'Am 29818 fournit les instructions que le séquenceur doit exécuter. Pour ce faire, ce dernier accède aux états nécessaires des Prom à registres. Le contrôle de l'ensemble du processeur périphérique peut se faire par l'observation des sorties des Prom à registres. Les tampons du canal d'entrée/sortie comprennent des Am 29818 pour les tests.

Cette architecture à bus unique simplifie les tests de diagnostic. Une manière directe d'effectuer les tests du bus consiste à fournir les configurations nécessaires pour vérifier les défaillances en blocage sur 1, sur 0 et les court-circuits par l'intermédiaire des registres à diagnostic du tampon d'E/S, et d'utiliser le registre de commande pour l'observation des résultats.

Les essais de diagnostic du séquenceur et de la Prom à registres sont effectués en chargeant en série l'adresse Prom désirée dans le registre espion du registre de commande qui fournit cette adresse au séquenceur, lequel la transmet à la Prom à registres. Les données accédées sont chargées dans le registre espion de la Prom à partir de son registre pipeline. Les résultats sont ensuite envoyés à l'extérieur et analysés par le processeur de diagnostic. Malheureusement, ce procédé ne permet pas de localiser une défaillance au niveau de la Prom ou du séquenceur.

Afin de pouvoir le faire il faut ajouter deux Am 29818 à la sortie du séquenceur. On peut ainsi court-circuter le séquenceur. En outre, la sortie de ce dernier peut être échantillonnée à cet endroit, assurant ainsi le test.

Un problème similaire se pose pour le test du circuit Am 2901 C commandant la Ram. On ne peut pas facilement accéder à l'adresse de cette dernière. En incorporant des Am 29818 au niveau du port d'adresse de la Ram, on obtiendra le point de test voulu. Maintenant, cette adresse peut être échantillonnée à partir de l'Am 2901 C au moment de l'essai du microprocesseur ou peut être fournie aux Am 9168 lors du test des Ram.

L'utilisation des Am 9168 dans les tampons de canaux E/S permet le contrôle de la liaison avec le périphérique. Il est souvent possible de placer le périphérique dans un mode en retour de boucle dans lequel les données transmises sont renvoyées vers l'émetteur. Cette technique peut être utilisée pour vérifier l'intégrité de la liaison entre le périphérique et le processeur périphérique.

Une fois que les cartes du processeur central et du processeur périphérique sont entièrement testées, on peut utiliser tous ces éléments afin de vérifier l'interface du bus. On utilise une carte pour transmettre des données de diagnostic à l'interface, et l'autre sert à échantillonner les résultats.

A propos des performances

Le problème de la répartition du contrôle des boucles de diagnostic peut avoir des conséquences sur les performances du système de diagnostic. Pour obtenir des performances et une souplesse maximales, il est préférable d'avoir des boucles courtes et localisées. En général, des boucles longues ont besoin d'un grand nombre de décalages pour obtenir des données de test significatives. Les boucles courtes facilitent par contre l'analyse des données, les longues chaînes de données étant difficiles à manipuler. Puisqu'on peut appliquer la même méthode de test à plusieurs unités fonctionnelles du système, on peut souvent utiliser un seul élément de code plusieurs fois, une fois pour chaque boucle courte.

Une boucle incorporée à chaque registre du système ne sera probablement pas utilisée pour tester le système en une fois, et sera donc probablement moins efficace. Néanmoins, un grand nombre de boucles de diagnostic courtes a tendance à compliquer le contrôle. Il faudra beaucoup plus de lignes DCLK, Mode, SDI, et SDO.

Le diagnostic SSR est une technique simple, facilement mise en œuvre et néanmoins extrêmement efficace, pour assurer le niveau de test désiré pour les systèmes informatiques à haute fiabilité. Elle permet l'accès à des points de test intermédiaires, en donnant la possibilité aux concepteurs de contrôler et d'observer toutes les unités fonctionnelles du système. Les codes de diagnostic peuvent ainsi être stockés en un endroit précis de la carte processeur de diagnostic. Comme de nombreux tests pour processeurs différents sont identiques, il est probable qu'on utilisera moins de code avec le diagnostic SSR. Le registre pipeline de diagnostic SSR Am 29818 et les Prom à registres de diagnostic de la série Am 27S65/75/85 sont des pièces qui fournissent aux concepteurs de systèmes une méthode économique de test.

Kevin Ow-Wing

RAM's on-chip registers build simple control stores that include self-diagnostics

Integrating a pipeline register and a shift register with a 4-bit RAM aids the design of writable control stores and yields access for built-in testing.

The growing popularity of microprogrammed computers has led to a need for more efficient and flexible control stores. A control store that uses RAM in place of ROM or PROM furnishes both efficiency and flexibility. Furthermore, a pipeline register speeds operation. Designers, however, can take another step to improve the store: Add a serial shift register to allow for self-diagnostics.

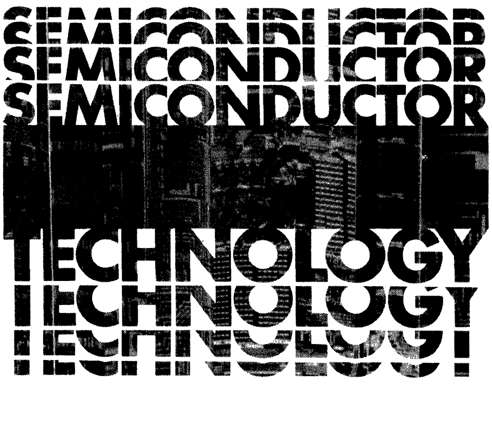
Unfortunately, large amounts of support circuitry have been necessary to implement a writable control store. Fortunately, one chip now combines a 4-bit static RAM with a pipeline register and a shift register and holds the required support circuits to a bare minimum.

The integration of the pipeline register and the RAM produces the typical benefits of lower chip count, faster operation, greater reliability, easier development, and simpler manufacturing. The shift register allows the use of a systematic diagnostic technique, known as serial-scan diagnostics, to pinpoint hardware failures in digital systems, thus making them simple to test and usually more reliable.

Bruce Threewitt, Advanced Micro Devices Inc.
Bruce Threewitt, manager of product planning for memories and memory support, has worked at Advanced Micro Devices in Sunnyvale, Calif., since 1981. After graduating with a BSEE from Oregon State University in 1970, he worked in applications engineering at Signetics and in applications engineering and strategic marketing at Fairchild.

Specifically designed to implement the writable control store of a bit-slice CPU, the Am9151 is a static n-channel device that contains a 1024-by-4-bit read/write memory, a parallel pipeline register, and a serial shadow shift register (Fig. 1). Available with minimum cycle times of 40, 50, or 60 ns, it operates from a single 5-V supply, has TTL input and output levels, and comes in a standard slim 24-pin dual in-line package.

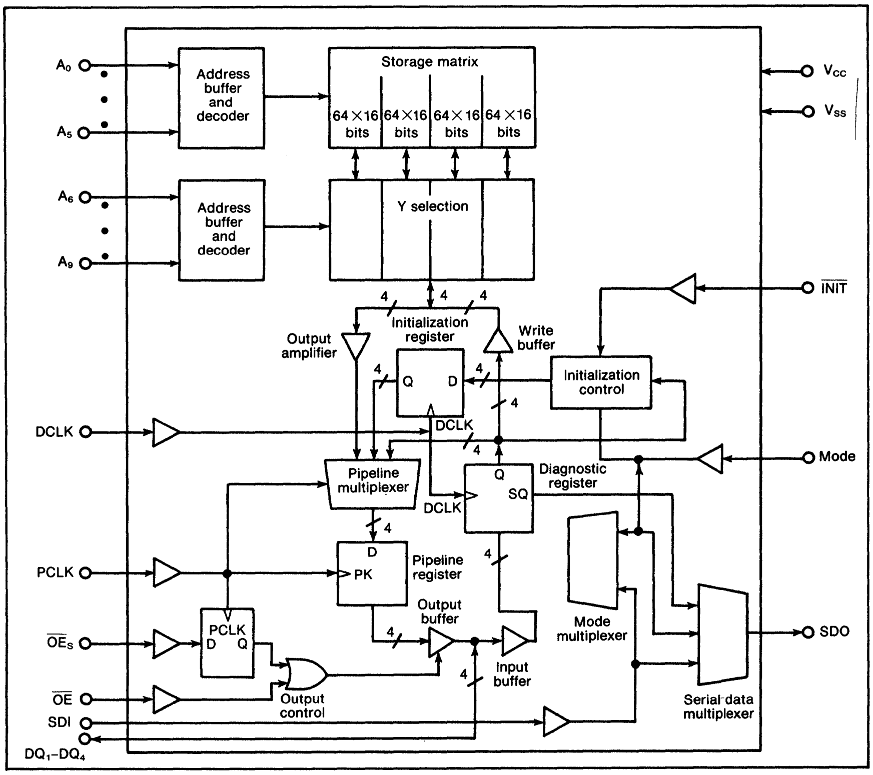
The chip joins a group of devices—a pipeline register chip and a family of PROMs with onboard registers—that include circuitry for implementing a form of serial-scan diagnostics called Serial Shadow Register (SSR) diagnostics (ELECTRONIC DESIGN, April 14, 1983, p.



119). Although the devices perform different functions, the SSR circuitry on each works in the same way. The algorithms needed to test different function blocks within a system are therefore consistent, simplifying system design. Also, the SSR diagnostic bus is a single line, simplifying design further.

The Serial Shadow Register can be used to examine and modify the contents of the parallel register and the memory. Consequently, replacing standard memory devices with the 9151 permits simple system verification and diagnostics. Moreover, connecting the serial output lines of several SSR parts in a row yields a serial loop with minimal interconnections and overhead hardware, which means that low-cost diagnostics can be built into other sections of a computer system.

A typical writable control store with serial-scan diagnostics includes RAM, a multiplexer, a serial shift register, a pipeline register, output buffers, a controller, a parallel data bus and serial I/O ports (Fig. 2). The shift register can be loaded from the serial input line or from



1. The Am9151 combines a parallel pipeline register with a 1024-by-4-bit static RAM to simplify the design of writable control stores. The on-board serial shift register makes it possible to modify and examine the contents of the memory and the pipeline register.

the bus, and shift register data can be transferred to the memory, the pipeline register, or the serial output port. The multiplexer controls the data path to the pipeline register, and the entire unit is controlled by clock edges.

Pin-count constraints can limit the width of WCS RAMs to 4 bits, but the store can be widened by connecting serial lines between the shift registers of consecutive units. The WCS is loaded or modified by writing data from the serial input into the serial shift register, from which it is moved into the memory in one clock cycle. A dedicated write-enable pin is not needed for the job.

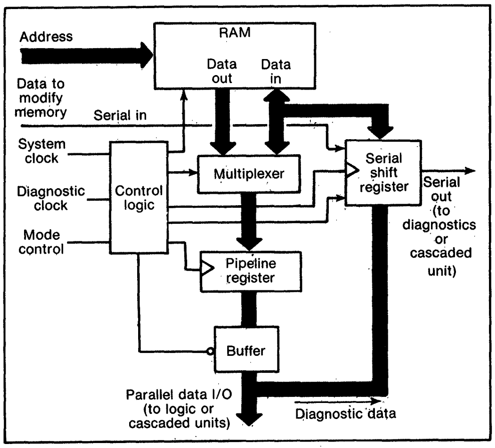
Control stores are usually read far more often than they are written. When the memory data is stable, it is clocked into the pipeline register. A new address can be applied to the device without affecting the contents of the pipeline register. The machine executes the microinstruction in the pipeline register while the WCS fetches the next microinstruction from memory. The memory address is always one clock cycle ahead of the pipeline register.

Diagnostics at two levels

A WCS that incorporates a Serial Shadow Register provides for diagnostics of the WCS's components and of the store itself. The components of the store can be checked through the shift register: Data is shifted through the serial ports of the shift register to test the register's integrity, then loaded through the pipeline register back into the shift register and to the serial output. The process is repeated for memory. If the output data differs from the input data, the component being tested is faulty.

WCS-level diagnostics can be initiated by the pipeline register, which checks and intercepts the instructions leaving the store. If an instruction is found to be wrong, the correct one can be shifted in serially to permit checking of the rest of the machine, provided that the remainder of the machine uses scan diagnostics. Another pipeline register can verify the result of a microinstruction and check and control feedback to the sequencer. Linking this pipeline register to the store's pipeline register allows the control logic to be examined during WCS diagnostics.

Sequence or diagnostic codes from a diagnostic unit can be loaded through the shift register, eliminating the need to store extra diagnostic codes in the control store. A floppy-disk-based diagnostic unit can also load microcode into the WCS memory. Diagnostics can even be performed from a remote location over ordinary telephone lines, using a modem to communicate with the system's central processing unit or diagnostics processor.



2. A serial shift register and appropriate control logic optimizes a writable control store for diagnostics.

In general, combinatorial logic networks create relatively minor testing problems. Sets of input signals can be applied to the network and the outputs compared with a set of calculated outputs to check for proper operation. The task can be minimized with computer-aided design tools, and fault coverage analysis can be automated to provide a measure of how efficient a set of test vectors is at pinpointing hardware failures.

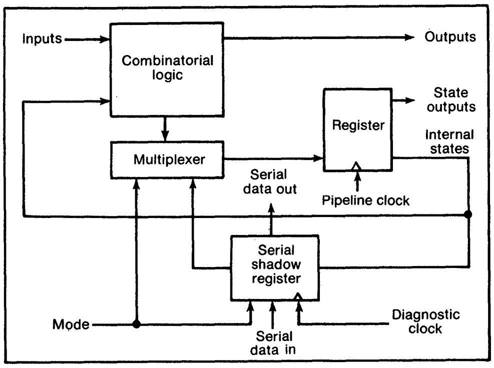
A sequential network like a computer system, on the other hand, is much more difficult to test systematically. The outputs of a sequential network depend not only on the present inputs but also on the internal state of the network. These are the kinds of circuits for which a logic analyzer is usually needed so that states that lead up to a problem can be examined. But even a logic analyzer cannot uncover logic states within a device if there is no output pin corresponding to the point that needs to be examined. Nor can it break feedback paths to check the forward characteristics of the sequential network.

Initializing the internal state register to the value necessary to test a given set of inputs is usually difficult, and observing the internal state of a sequential network can be both difficult and time-consuming if state information is not directly available. For example, to determine the value of an internal 16-bit counter for which only the carry-out signal is accessible requires that the counter be clocked until it reaches the carry-out state and the starting value computed. Up to 65,536 clock cycles may be necessary.

Sequential into combinatorial

The Serial Shadow Register provides a better method. By establishing the means to initialize and sample the state elements of a sequential network (Fig. 3), it turns a sequential network into a combinatorial one.

An extra multiplexer on the input of each state register and a duplicate or shadow of each state flip-flop in an additional register are at the heart of the technique. The shadow register can be loaded serially via the serial



3. The diagnostic shadow register can be used to initialize and sample internal state registers, turning difficult-to-test sequential logic systems into easier-to-test combinatorial logic systems.

data input for controllability. Once loaded into the serial register, the desired state information can be transferred to the internal state register by selecting the multiplexer and clocking the state register. That allows any internal point to be set to a desired state simply, quickly, and systematically.

Internal state information is sampled by loading the serial register from the state register. This information is then shifted out via the serial data output so that it can be observed. The serial data inputs and outputs can be cascaded to make long chains of state information available with a minimum number of connections.

In effect, SSR diagnostics breaks the normal feedback path of a sequential network and establishes a logical path with which inputs can be defined and outputs sampled. As a result, techniques that have been developed to test combinatorial networks can be applied to a sequential network in which SSR diagnostics are implemented.

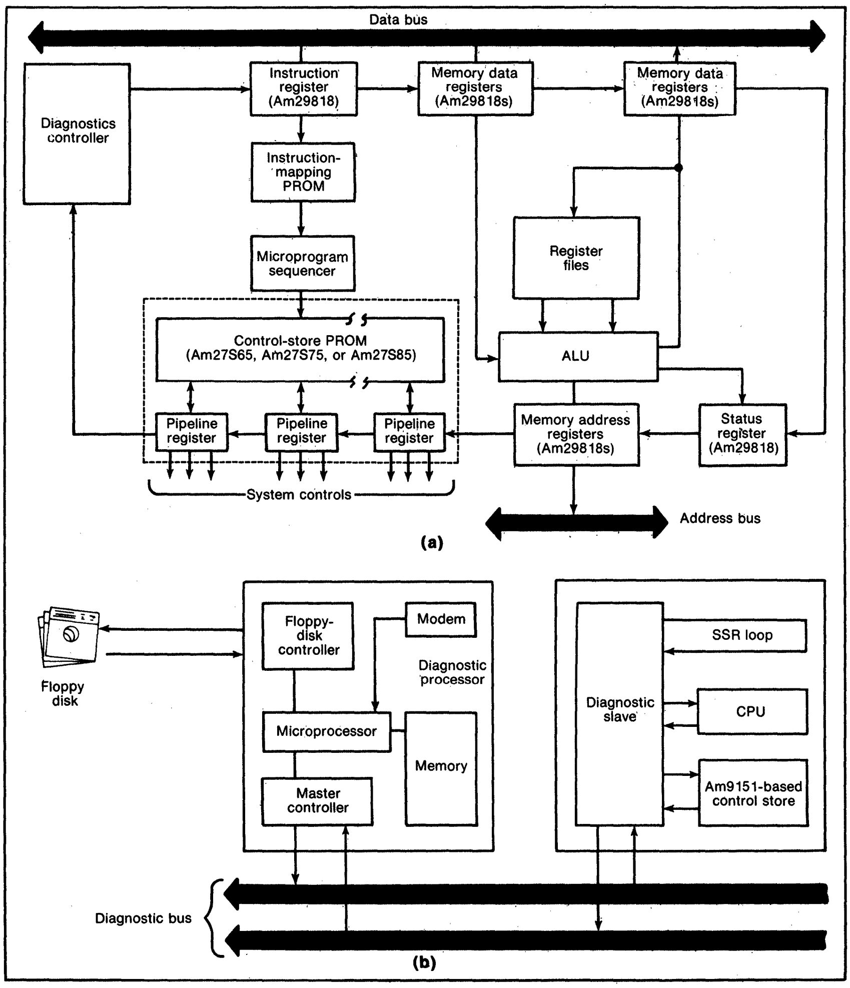
Taking advantage of diagnostics

In a microprogrammable computer that takes advantage of SSR diagnostics, a serial path is added to all important state registers—macroinstruction, data, status, address, and microinstruction (Fig. 4a). The extra path makes it easier to diagnose system failures by breaking the feedback paths and turning sequential state machines into combinatorial logic blocks. For example, the status outputs of an ALU may be checked by loading the microinstruction register with the necessary data. The desired ALU function is then executed and the status outputs captured in the status register.

The status bits can be shifted out serially and checked for accuracy. Several loops can be employed in complicated systems to reduce scan time, and a single diagnostic processor controls all test processing and provides input and output to telephone lines via a modem and from a floppy disk (Fig. 4b).

Now, large amounts of support circuitry are necessary to implement a writable control store. Additional input and output buffers are required to furnish paths from the parallel-input data bus to the memory and from the instruction register to the output data bus. An input port is necessary to write data to the control store, initializing the microinstruction memory, and an output port provides access to the instruction register, indirectly allowing the RAM to be read. Additionally, access to the instruction register is useful during system debugging and diagnostics.

The 9151 handles all these operations and more with minimal additional circuitry. A typical WCS design built around several of the chips (Fig. 5) provides access to memory and random-logic states over a serial diagnostic port. The instruction register's contents can



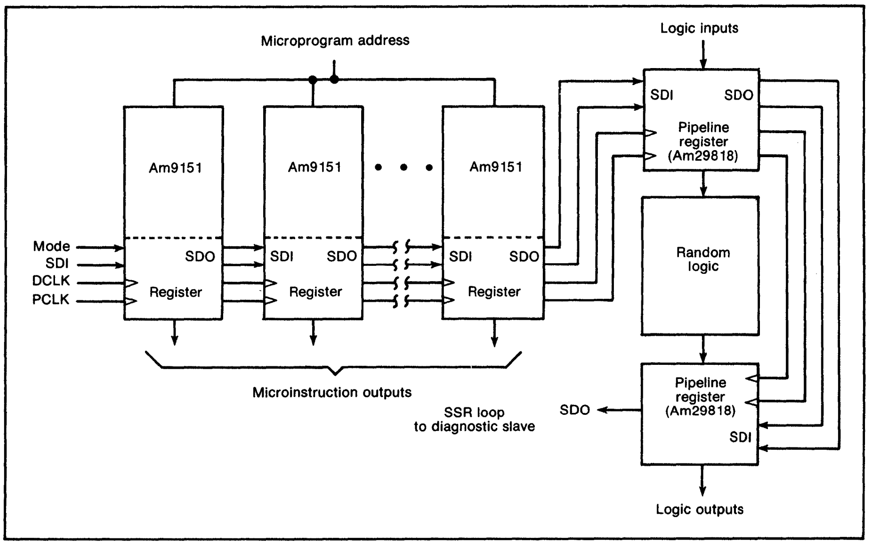
4. If SSR diagnostic hardware is incorporated and placed properly in a typical computer system (a), the system gains the ability to individually test each functional unit. An inexpensive diagnostic processor (b) can read data from a floppy disk for routine preventive diagnostic test maintenance. Field service engineers can diagnose failures and determine the failing components over the phone lines.

Semiconductor Technology: 4k RAM for writable control stores

be read by serially shifting the information out through the diagnostics port. In addition, the instruction register can be written from the serial port via the shadow register, considerably simplifying system debugging and diagnostic operations.

To load a microcode word into the shadow register from the pipeline register, for example, requires simply that the Serial Data Input line (SDI) be held low and the Mode pin held high. The transfer takes place when the Diagnostic Clock input (DCLK) switches from low to high. Pulling the Mode pin low then shifts data out of the shadow register on further DCLK transitions from low to high. Diagnostic information is shifted serially through the single-line diagnostic loop to the diagnostic processor, where data can be checked for accuracy.

Loading data into the writable control store via the shadow register is equally direct. Code from the diagnostic processor, whether original code to initialize the system, test vectors designed to check the operation of the system, or new code that changes the characteristics of the system, is shifted serially into the shadow register. When the Mode pin goes high, data from the shadow register is transferred into the pipeline register in parallel on the transition of the Pipeline Clock signal (PCLK) from low to high. Data held in the diagnostic registers is loaded into memory when the SDI and Mode pins are held high and the DCLK pin switches from low to high. The process of loading data into memory via the shadow register is relatively slow because of the serial shift from the diagnostic processor. But this activity is not needed often—only at system initialization or during diagnostics, for example, and not during normal system operation.□



5. The serial diagnostic loop provides a standard mechanism by which state information can be controlled and observed, simplifying diagnostic operations and making it possible to pinpoint failures at the component level.

World-Wide Sales Offices Representatives Advanced Micro Devices

U.S. AND CANADIAN SALES OFFICES

NORTHEAST AREA

Advanced Micro Devices
20 Mall Road
Burlington, Massachusetts 01803
Tel: (617) 273-3970

Advanced Micro Devices (Canada) Ltd.
2 Sheppard Avenue East
Suite 1610
Willowdale, Ontario
Canada M2N5Y7
Tel: (416) 224-5193

Advanced Micro Devices (Canada) Ltd.
AMD
4019 Carling #301
Kanata, Ottawa
Canada K2K2A3
Tel: (613) 592-0060

Advanced Micro Devices
290 Elwood Davis Road
Suite 316
Liverpool, New York 13088
Tel: (315) 457-5400

MID ATLANTIC AREA

Advanced Micro Devices
Gateways
1000 Woodbury Road
Woodbury, New York 11797
Tel: (516) 364-8020

Advanced Micro Devices
Waterview Plaza, Suite 303
2001 U.S. Route #46
Parsippany, New Jersey 07054
Tel: (201) 299-0002

Advanced Micro Devices
2300 Computer Avenue
Suite D 19
Willow Grove, Pennsylvania 19090
Tel: (215) 657-3101

Advanced Micro Devices
Commerce Plaza
5100 Tilghman Street, Suite 320
Allentown, Pennsylvania 18104
Tel: (215) 398-8006

Advanced Micro Devices
205 South Avenue
Poughkeepsie, New York 12601
Tel: (914) 471-8180

Advanced Micro Devices
10 Main Street South
Southbury, Connecticut 06488
Tel: (203) 264-7800

Advanced Micro Devices
7223 Parkway Drive #203
Dorsey, Maryland 21076
Tel: (301) 796-9310

SOUTHEAST AREA

Advanced Micro Devices
4740 North State Road #7
Bldg. C—Suite 202
Ft. Lauderdale, Florida 33319
Tel: (305) 484-8600

Advanced Micro Devices
15351 Roosevelt Boulevard #201
Clearwater, Florida 33520
Tel: (813) 530-9971

Advanced Micro Devices
478 Ballard Drive #14
Melbourne, Florida 32935
Tel: (305) 254-2915

Advanced Micro Devices
701 East Altamonte Drive
Suite 204
Altamonte Springs, Florida 32701
Tel: (305) 834-3333

Advanced Micro Devices
15 Technology Parkway #200
Norcross, Georgia 30092
Tel: (404) 449-7920

Advanced Micro Devices
8 Woodlawn Green, Suite 220
Woodlawn Road
Charlotte, North Carolina 28210
Tel: (704) 525-1875

Advanced Micro Devices
500 Boulevard South
Suite 201
Huntsville, Alabama 35802
Tel: (205) 882-9122

Advanced Micro Devices
6501 Six Forks, Suite 150
Raleigh, North Carolina 27609
Tel: (919) 847-8471

MID AMERICA AREA

Advanced Micro Devices
500 Park Boulevard, Suite 940
Itasca, Illinois 60143
Tel: (312) 773-4422

Advanced Micro Devices
5726 Professional Circle #205G
Indianapolis, Indiana 46241
Tel: (317) 244-7207

Advanced Micro Devices
7007 College Blvd.
Suite 330
Overland Park, Kansas 66211
Tel: (913) 451-3115

Advanced Micro Devices
9900 Bren Road East, Suite 601
Minnetonka, Minnesota 55343
Tel: (612) 938-0001

Advanced Micro Devices
3592 Corporate Drive, Suite 108
Columbus, Ohio 43229
Tel: (614) 891-6455

Advanced Micro Devices
16985 West Blue Mound Road, Suite 201
Brookfield, Wisconsin 53005
Tel: (414) 782-7748

NORTHWEST AREA

Advanced Micro Devices
1245 Oakmead Parkway
Suite 2900
Sunnyvale, California 94086
Tel: (408) 720-8811

Advanced Micro Devices
One Lincoln Center, Suite 230
10300 Southwest Greenburg Road
Portland, Oregon 97223
Tel: (503) 245-0080

Advanced Micro Devices
Honeywell Ctr., Suite 1002
600 108th Avenue N.E.
Bellevue, Washington 98004
Tel: (206) 455-3600

MID-CALIF AREA

Advanced Micro Devices
360 N. Sepulveda, Suite 2075
El Segundo, California 90245
Tel: (213) 640-3210

Advanced Micro Devices
21600 Oxnard Street, Suite 675
Woodland Hills, California 91367
Tel: (818) 992-4155

SOUTHERN CALIF AREA

Advanced Micro Devices
5000 Birch Street
Suite 6000
Newport Beach, California 92660
Tel: (714) 752-6262

Advanced Micro Devices
9619 Chesapeake Drive #210
San Diego, California 92123
Tel: (619) 560-7030

MOUNTAIN WEST AREA

Advanced Micro Devices
14755 Preston Road, Suite 700
Dallas, Texas 75240
Tel: (214) 934-9099

Advanced Micro Devices
9020 Capital of Texas
Highway North, Suite 345
Austin, Texas 78759
Tel: (512) 346-7830

Advanced Micro Devices
2925 Briar Park #410
Houston, Texas 77042
Tel: (713) 785-9001

Advanced Micro Devices
1873 South Bellaire Street
Suite 920
Denver, Colorado 80222
Tel: (303) 691-5100

Advanced Micro Devices
40 W. Baseline Road #206
Tempe, Arizona 85283
Tel: (602) 242-4400

Advanced Micro Devices
1955 W. Grant Road #125
Tucson, Arizona 85745
Tel: (602) 792-1200

INTERNATIONAL SALES OFFICES

BELGIUM
Advanced Micro Devices
Belgium S.A.—N.V.
Avenue de Tervueren, 412, bte 9
B-1150 Bruxelles
Tel: (02) 771 99 93
TELEX: 61028
FAX: 7623712

FRANCE
Advanced Micro Devices, S.A.
Silic 314, Immeuble Helsinki
74, rue d'Arcueil
F-94588 Rungis Cedex
Tel: (01) 687.36.66
TELEX: 202053
FAX: 686.21.85

GERMANY
Advanced Micro Devices GmbH
Rosenheimer Strasse 143B
D-8000 München 80,
West Germany
Tel: (089) 4114-0
TELEX: 05-23883
FAX: 406 490

GERMANY
Advanced Micro Devices GmbH
Feuerseeplatz 4/5
D-7000 Stuttgart 1
Tel: (0711) 62 33 77
TELEX: 07-21882
FAX: 625 187

Advanced Micro Devices GmbH
Wünning Weg 4
D-3108 Winsen/Aller
Tel: (05143) 5055
TELEX: 925287
FAX: (05143) 5553

HONG KONG
Advanced Micro Devices
Room 1602 World Finance Centre
South Tower
Harbour City
17 Canton Road
Tsimshatsui, Kowloon
Tel: (852) 3 695377
TELEX: 50426
FAX: (852) 123 4276

ITALY
Advanced Micro Devices S.R.L.
Centro Direzionale
Via Novara, 570
I-20153 Milano
Tel: (02) 3390541
TELEX: 315286
FAX: (39) 349 8000

JAPAN
Advanced Micro Devices, K.K.
Shinjuku Kokusai Building
6-6-2 Nishi-Shinjuku
Shinju-ku, Tokyo 160
Tel: (03) 345-8241
TELEX: 24064
FAX: 03 (342) 5196

SWEDEN
Advanced Micro Devices AB
Box 2028
Rissneleden 144, 5tr
S-172 02 Sundbyberg
Tel: (08) 733 03 50
TELEX: 11602
FAX: 7332285

UNITED KINGDOM
Advanced Micro Devices (U.K.) Ltd.
A.M.D. House,
Goldsworth Road,
Woking,
Surrey GU21 1JT
Tel: Woking (04862) 22121
TELEX: 859103
FAX: 22179

Advanced Micro Devices (U.K.) Ltd.
The Genesis Centre
Garrett Field
Science Park South
Birchwood
Warrington WA3 7BH
Tel: Warrington (0925) 828008
TELEX: 628524
FAX: 827693

U.S. AND CANADIAN SALES REPRESENTATIVES

CALIFORNIA (Northern)
I^2 Incorporated
3350 Scott Boulevard
Suite 1001, Bldg. 10
Santa Clara, California 95050
Tel: (408) 988-3400
TWX: 910-338-0192

CANADA (Eastern)
Vitel Electronics
3300 Cote Vertu, Suite 203
St. Laurent, Quebec,
Canada H4R 2B7
Tel: (514) 331-7393
TWX: 610-421-3124
TELEX: 05-821762

Vitel Electronics
5945 Airport Road, Suite 180
Mississauga, Ontario
Canada L4V 1R9
Tel: (416) 676-9720
TWX: 610-492-2528

Vitel Electronics
4019 Carling #301
Kanata, Ottawa
Canada K2K 2A3
Tel: (613) 836-1776

CONNECTICUT
Scientific Components
1185 South Main Street
Cheshire, Connecticut 06410
Tel: (203) 272-2963
TWX: 710-455-2078

IDAHO
Intermountain Technology
1106 N. Cole Road #C
Boise, Idaho 83704
Tel: (208) 888-5708

INDIANA
S.A.I. Marketing Corp.
2441 Production Drive
Indianapolis, Indiana 46241
Tel: (317) 241-9276
TWX: 810-341-3309

IOWA
Lorenz Sales, Inc.
5270 N. Park Place, N.E.
Cedar Rapids, Iowa 52402
Tel: (319) 377-4666

KANSAS
Kebco Inc., c/o Doug Phillips
524 Deveron Drive
Wichita, Kansas 67230
Tel: (316) 733-2117
(316) 733-1301

Kebco Inc.
10111 Santa Fe Avenue, Suite 13
Overland Park, Kansas 66212
Tel: (913) 541-8431

MICHIGAN
S.A.I. Marketing Corp.
P.O. Box 929
9880 E. Grand River Road #109
Brighton, Michigan 48116
Tel: (313) 227-1786
TWX: 810-242-1518

MISSOURI
Kebco Inc.
75 Worthington, Suite 101
Maryland Heights, Missouri 63043
Tel: (314) 576-4111

NEBRASKA
Lorenz Sales, Inc.
2809 Garfield
Lincoln, Nebraska 68502
Tel: (402) 475-4660

NEW JERSEY
T.A.I. Corp.
12 S. Black Horse Pike
Bellmawr, New Jersey 08031
Tel: (609) 933-2600
TWX: 710-639-1810

NEW MEXICO
Thorson Desert States
9301 Indian School, Suite 112
Albuquerque, New Mexico 87112
Tel: (505) 293-8555
TWX: 910-989-1174

NEW YORK
Nycom, Inc.
10 Adler Drive
East Syracuse, New York 13057
Tel: (315) 437-8343
TWX: 710-541-1506

OHIO
Dolfuss-Root & Co.
13477 Prospect Road
Strongsville, Ohio 44136
Tel: (216) 238-0300
TWX: 810-427-9148

Dolfuss-Root & Co.
683 Miamisburg-Centerville Road
Suite 202
Centerville, Ohio 45459
Tel: (513) 433-6776

PENNSYLVANIA
Dolfuss-Root & Co.
United Industrial Park
Suite 203A, Building A
98 Vanadium Road
Bridgeville, Pennsylvania 15017
Tel: (412) 221-4420
TWX: 510-697-3233

UTAH
R^2 Marketing
2070 North Redwood Road
Salt Lake City, Utah 84116
Tel: (801) 595-0631

U.S. AND CANADIAN DISTRIBUTORS

ALABAMA

Arrow Electronics
1015 Henderson Road
Huntsville, Alabama 35805
Tel: (205) 837-6955

Hall-Mark Electronics
4900 Bradford Boulevard
Huntsville, Alabama 35807
Tel: (205) 837-8700
TWX: 810-726-2187

Hamilton/Avnet Electronics
4812 Commercial Drive
Huntsville, Alabama 35805
Tel: (205) 837-7210
TWX: 810-726-2162

Schweber Electronics
2227 Drake Avenue S.W.
Suite #14
Huntsville, Alabama 35805
Tel: (205) 882-2200

ARIZONA

Arrow Electronics
2127 W. 5th Place
Tempe, Arizona 85281
(602) 968-4800

Hamilton/Avnet Electronics
505 South Madison Drive
Tempe, Arizona 85281
Tel: (602) 231-5100
TWX: 910-951-1535

Kierulff Electronics
4134 East Wood Street
Phoenix, Arizona 85040
Tel: (602) 243-4101
TWX: 910-951-1550

Schweber Electronics
11049 North 23rd Drive
Phoenix, Arizona 85029
Tel: (602) 997-4874

Wyle Labs/EMG
8155 North 24th Avenue
Phoenix, Arizona 85021
Tel: (602) 249-2232
TWX: 910-951-4282

CALIFORNIA

Arrow Electronics
2961 Dow Avenue
Tustin, California 92680
Tel: (714) 838-5422
TWX: 910-595-2861

Arrow Electronics
9511 Ridgehaven Court
San Diego, California 92123
Tel: (619) 565-4800
TWX: 910-335-1195

Arrow Electronics
19748 Dearborn Street
Chatsworth, California 91311
Tel: (213) 701-7500
TWX: 910-493-2086

Arrow Electronics
521 Weddell Drive
Sunnyvale, California 94086
Tel: (408) 745-6600
TWX: 910-339-9371

Arrow Electronics
1808 Tribute Road, Suite C
Sacramento, California 95815
Tel: (916) 925-7456

Arrow Electronics
1502 Crocker Avenue
Hayward, California 94544
Tel: (415) 487-4600

Avnet Electronics
350 McCormick Avenue
Irvine Industrial Complex
Costa Mesa, California 92626
Tel: (714) 754-6111
(213) 558-2345
TWX: 910-595-1928

Avnet Electronics Co. #71
20501 Plummer Street
Chatsworth, California 91311
Tel: (213) 700-2600

Arrow Electronics
1502 Crocker Avenue
Hayward, California 94544
Tel: (415) 487-4600

Hamilton/Avnet Electronics
3170 Pullman
Costa Mesa, California 92626
Tel: (714) 641-4100
TWX: 910-595-2638

Hamilton/Avnet Electronics
3002 East G Street
Ontario, California 91764
Tel: (714) 989-9411

Hamilton Electro Sales
9650 Desoto Avenue
Chatsworth, California 91311
Tel: (818) 700-6500

Hamilton/Avnet Electronics
4103 Northgate Boulevard
Sacramento, California 95834
Tel: (916) 920-3150

Hamilton/Avnet Electronics
4545 View Ridge Road
San Diego, California 92123
Tel: (619) 571-7500
TWX: 910-335-1216

Hamilton/Avnet Electronics
1175 Bordeaux
Sunnyvale, California 94086
Tel: (408) 743-3300
TWX: 910-339-9332

Hamilton Electro Sales
10950 West Washington Boulevard
Culver City, California 90230
Tel: (213) 558-2121
TWX: 910-340-6364
910-340-7073
TELEX: 66-43-29
66-43-31

Kierulff Electronics
5650 Jillson
Commerce, California 90040
Tel: (213) 725-0325

Kierulff Electronics
21053 Devonshire #203
Chatsworth, California
Tel: (818) 341-2211

Kierulff Electronics
10824 Hope Street
Cypress, California 90630
Tel: (714) 220-6300

Kierulff Electronics
1180 Murphy Road
San Jose, California 95131
Tel: (408) 751-6262

Kierulff Electronics
8797 Balboa Avenue
San Diego, California 92123
Tel: (619) 278-2112
TWX: 910-335-1182

Kierulff Electronics
14101 Franklin Avenue
Tustin, California 92680
Tel: (714) 731-5711

Schweber Electronics
17822 Gillette
Irvine, California 92714
Tel: (213) 537-4320
(714) 863-0200
TWX: 910-595-1720

Schweber Electronics
90 East Tasman
San Jose, California 95134
Tel: (408) 946-7171

Schweber Electronics
1771 Tribute Road
Suite B
Sacramento, California 95815
Tel: (916) 929-9732

Schweber Electronics
21139 Victory Boulevard
Canoga Park, California 91303
Tel: (213) 999-4702

Schweber Electronics
1225 West 190th Street
Gardena, California 90248
Tel: (213) 327-8409

Schweber Electronics
Carroll Ridge Business Center
6750 Nancy Ridge Road
Suite D&E, Bldg. 7
San Diego, California 92121
Tel: (619) 450-0454

Wyle Labs/EMG
26560 Agoura Road
Calabasas, California 91302
Tel: (818) 880-9000

Wyle Labs/EMG
124 Maryland Street
El Segundo, California 90245
Tel: (213) 332-8100

Wyle Labs/EMG
Orange County Division
17872 Cowan
Irvine, California 92714
Tel: (714) 863-9953
TWX: 910-595-1572

Wyle Labs/EMG
11151 Sun Center Drive
Rancho Cordova, California 95670
Tel: (916) 638-5282

Wyle Labs/EMG
9525 Chesapeake Drive
San Diego, California 92123
Tel: (619) 565-9171
TWX: 910-335-1590

Wyle Labs/EMG
3000 Bowers Avenue
Santa Clara, California 95052
Tel: (408) 727-2500
TWX: 910-338-0296

CANADA

Arrow/Cesco
4050 Jean Talon Ouest
Montreal, Quebec
Canada H4P 1W1
Tel: (514) 735-5511
TWX: 05-255-90

Arrow/Cesco
24 Martin Ross Avenue
Downsview, Ontario
Canada M3J 2K9

Arrow/Cesco
148 Colonnade Road
Nepean, Ontario
Canada K2E 7J5
Tel: (613) 226-6903

Arrow/Cesco
909 Boul. Charet Ouest
Quebec, Quebec
Canada G1N 2C9
Tel: (418) 687-4231

Future Electronics
82 St. Regis Crescent North
Downsview, Ontario
Canada M3J 1Z3
Tel: (416) 663-5563
TWX: 610-491-1470

Future Electronics
Baxter Centre
1050 Baxter Road
Ottawa, Ontario
Canada K2C 3P2
Tel: (613) 820-8313
TWX: 610-563-1697

Future Electronics
237 Hymus Boulevard
Pointe Claire, Quebec, Canada H9R 5C7
Tel: (514) 694-7710
TWX: 610-421-3251
TELEX: 82-3554
05-823555

Future Electronics
3070 Kingsway
Vancouver, British Columbia
Canada V5R 5J7
Tel: (604) 438-5545
TWX: 610-922-1668

Hamilton/Avnet Electronics
6845 Rexwood Road, Units 3-5
Mississauga, Ontario, Canada L4V 1M5
Tel: (416) 677-7432
TWX: 610-492-8867

Hamilton/Avnet Electronics
210 Colonnade
Neaton, Ontario, Canada K2E 7L5
Tel: (613) 226-1700
TWX: 610-562-1906

Hamilton/Avnet Electronics
2816 21st Street, N.E.
Calgary, Alberta, Canada T2E 6Z2
Tel: (403) 230-3586
TWX: 610-821-2286

Hamilton/Avnet Electronics
2670 Sapourin
St. Laurent, Quebec, Canada H4S 1M2
Tel: (514) 331-6443
TWX: 610-421-3731

Hamilton/Avnet Electronics
2250 Boundary Road
Burnaby, British Columbia
Canada V5M 3Z3
Tel: (604) 272-4242

COLORADO

Arrow Electronics
1390 South Potomac Street
Suite 136
Aurora, Colorado 80012
Tel: (303) 696-1111
TWX: 910-932-2999

Hamilton/Avnet Electronics
8765 East Orchard Road
Suite 708
Englewood, Colorado 80111
Tel: (303) 740-1000
TWX: 910-935-0787

Kierulff Electronics
7060 South Tuscon Way
Englewood, Colorado 80112
Tel: (303) 790-4444
TWX: 910-931-2626

Schweber Electronics
Highland Technical Business Park
8955 East Nichols Avenue
Suite 200
Englewood, Colorado 80112
Tel: (303) 799-0258

Wyle Labs/EMG
451 East 124th Avenue
Thornton, Colorado 80241
Tel: (303) 457-9953
TWX: 910-936-0770

CONNECTICUT

Arrow Electronics
12 Beaumont Road
Wallingford, Connecticut 06492
Tel: (203) 265-7741
TWX: 710-476-0162

Hamilton/Avnet Electronics
Commerce Park
Commerce Drive
Danbury, Connecticut 06810
Tel: (203) 797-2800
TWX: 710-456-9974

Schweber Electronics
Finance Drive
Commerce Industrial Park
Danbury, Connecticut 06810
Tel: (203) 748-7080
TWX: 710-456-9405

FLORIDA

Arrow Electronics
4902 Creekside Drive
Suite A
Clearwater, Florida 33526
Tel: (813) 576-8995

Arrow Electronics
1530 Bottlebrush Drive
Palm Bay, Florida 32905
Tel: (305) 725-1480
TWX: 510-959-6337

Arrow Electronics
350 Fairway Drive
Deerfield Beach, Florida 33441
Tel: (305) 429-8200

Arrow Electronics
Eastern Building
Suite 503
DeDiego Street
Santurce, Puerto Rico 00911

Hall-Mark Electronics
15301 Roosevelt Boulevard
Suite 303
Clearwater, Florida 33520
Tel: (813) 530-4543

Hall-Mark Electronics
3161 S.W. 15th Street
Pompano Beach, Florida
33069-4806
Tel: (305) 971-9280
TWX: 510-956-9720

Hall-Mark Electronics
7648 Southland Boulevard
Suite 100
Orlando, Florida 32809
Tel: (305) 855-4020
TWX: 810-850-0183

Hamilton/Avnet Electronics
6801 N.W. 15th Way
Ft. Lauderdale, Florida 33309
Tel: (305) 971-2900
TWX: 510-956-3097

Hamilton/Avnet Electronics
3197 Tech Drive North
St. Petersburg, Florida 33702
Tel: (813) 576-3930
TWX: 810-863-0374

Hamilton/Avnet Electronics
6947 University Blvd.
Winter Park, Florida 32792
Tel: (305) 628-3888

Kierulff Electronics
3247 Tech Drive
St. Petersburg, Florida 33702
Tel: (813) 576-1966
TWX: 810-863-5625

Schweber Electronics
215 North Lake Boulevard
Altamonte Springs, Florida 32701
Tel: (305) 331-7555

Schweber Electronics
2830 North 28th Terrace
Hollywood, Florida 33020
Tel: (305) 927-0511

GEORGIA

Arrow Electronics
3155-A Northwoods Parkway
Norcross, Georgia 30092
Tel: (404) 449-8252
TWX: 810-766-0439

Hall-Mark Electronics
6410 Atlantic Boulevard, Suite 115
Norcross, Georgia 30071
Tel: (404) 447-8000
TWX: 810-766-4510

Hamilton/Avnet Electronics
5825-D Peachtree Crossing
Norcross, Georgia 30092
Tel: (404) 447-7500
TWX: 810-766-0432

Schweber Electronics
303 Research Drive
Suite 210
Norcross, Georgia 30092
Tel: (404) 449-9170
TWX: 810-766-1592

ILLINOIS

Arrow Electronics
2000 Algonquin
Schaumburg, Illinois 60195
Tel: (312) 397-3440

Hall-Mark Electronics
210 Mittel Drive
Wooddale, Illinois 60191
Tel: (312) 860-3800

Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville, Illinois 60106
Tel: (312) 860-7780
TWX: 910-227-0060

**ADVANCED
MICRO
DEVICES, INC.**
*901 Thompson Place
P.O. Box 3453
Sunnyvale,
California 94088
(408) 732-2400
TWX: 910-339-9280
TELEX: 34-6306
TOLL FREE
(800) 538-8450*

ISSN 0883-1262

06903A

CBM-MU-5M-6/85-0